CONTENTS

BLOCK III: - NORMALIZATION CONCEPTS & QUERY PROCESSING

UNIT I – DATABASE MANAGEMENT SYSTEM CONCEPTS

Structure

1.1. INTRODUCTION

As an end in itself, understanding database management system concepts and terms is important to enable you learn more about DBMS and attributes of DBMS after the course is over. Without understanding these concepts and terms, you will have difficulty discussing database management system ideas with others, and will have difficulty in reading the technical literature. Since computer science is rapidly evolving new database management system and since the issues are important in many areas of computer science the ability to learn more quickly is important to maintaining your technical edge.

1.2. OBJECTIVES

After going through this lesson you would be in a positions to

- Use ideas from the various paradigms of database that is not explicitly suited to that paradigm.
- Implement important run-time database management system and its applications, entities and its attributes, advantages and disadvantages and compare the DBMS and RDBMS that use such implementations.

1.3. DATABASE MANAGEMENT SYSTEM CONCEPTS

Data is nothing but facts and statistics stored or free flowing over a network, generally it's raw and unprocessed. For example: When you visit any website, they might store you IP address, that is data, in return they might add a cookie in your browser, marking you that you visited the website, that is data, your name, it's data, your age, it's data.

A Database is a collection of related data organised in a way that data can be easily accessed, managed and updated. Database can be software based or hardware based, with one sole purpose, storing data. During early computer days, data was collected and stored on tapes, which were mostly write-only, which means once data is stored on it, it can never be read again. They were slow and bulky, and soon computer scientists realized that they needed a better solution to this problem.

A DBMS is software that allows creation, definition and manipulation of database, allowing users to store, process and analyse data easily. DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more.

DBMS also provides protection and security to the databases. It also maintains data consistency in case of multiple users. It consists of a group of programs which manipulate the database and provide an interface between the databases. It includes the user of the database and other application programs.

The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data. In large systems, a DBMS helps users and other third-party software to store and retrieve data.

1.3.1. Significance of Database

One of the major aims of a database is to supply users with an abstract view of data, hiding a certain element of how data is stored and manipulated. So, the starting point for the

design of a database must be an abstract and general description of the information requirements of the organization that is to be represented in the database. And hence you will require an environment to store data and make it work as a database.

A database management system is important because it manages data efficiently and allows users to perform multiple tasks with ease. A database management system stores, organizes and manages a large amount of information within a single software application. Use of this system increases efficiency of business operations and reduces overall costs.

Database management systems are important to businesses and organizations because they provide a highly efficient method for handling multiple types of data. Some of the data that are easily managed with this type of system include: employee records, student information, payroll, accounting, project management, inventory and library books. These systems are built to be extremely versatile.

Without database management, tasks have to be done manually and take more time. Data can be categorized and structured to suit the needs of the company or organization. Data is entered into the system and accessed on a routine basis by assigned users. Each user may have an assigned password to gain access to their part of the system. Multiple users can use the system at the same time in different ways.

A simple database has a single table with rows for the data and columns that define the data elements. For an address book, the table columns define data elements such as name, address, city, state and phone number, while a table row, or record, contains data for each person in the book. The query language provides a way to find specific types of data in each record and return results that match the criteria. These results display in a form that uses the defined data elements but only shows records that meet the criteria. These three components make up almost every type of database.

Relational databases use multiple tables and define relationships between them using a schema in addition to data elements. Records and data elements from each table merge, based on the query, and display in the form. Routinely used queries often become reports. A report uses the same query but reports on changes in data over time.

1.3.2. Database System Applications

Applications where we use Database Management Systems are:

- Telecom: There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- Industry: Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.

- Banking System: For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.
- Sales: To store customer information, production information and invoice details.
- Airlines: To travel though airlines, we make early reservations, this reservation information along with flight schedule is stored in database.
- Education sector: Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.
- Online shopping: You must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

These are a very few applications, this list is never going to end if we start mentioning all the DBMS applications.

1.3.3. Data Independence

If a database system is not multi-layered, then it becomes difficult to make any changes in the database system. Database systems are designed in multi-layers as we learnt earlier.

A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database. But as a DBMS expands, it needs to change over time to satisfy the requirements of the users. If the entire data is dependent, it would become a tedious and highly complex job.

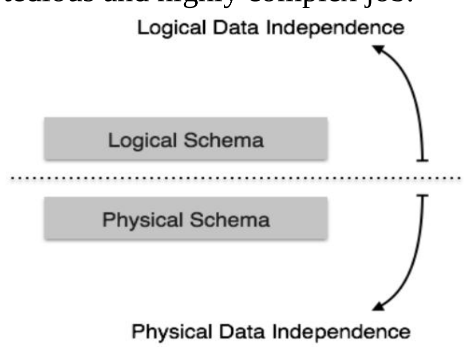


Figure 1: - Data Independence

Metadata itself follows a layered architecture, so that when we change data at one layer, it does not affect the data at another level. This data is independent but mapped to each other.

Logical Data Independence

Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints, applied on that relation.

Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk.

Physical Data Independence

All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data.

For example, in case we want to change or upgrade the storage system itself − suppose we want to replace hard-disks with SSD − it should not have any impact on the logical data or schemas.

1.3.4. Data Modeling for a Database

Data models define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system.

The data model plays an important role in database design. The physical or logical structure of a database is spelt out by the data model. A data model is a collection of conceptual tools used for describing data, data relationships, data semantics and data constraints. Evaluation of different data models is still in progress as the primary objective is to evolve a high level data model. The model should enable the designer to incorporate a major portion of semantics of the database in the schema. Numerous data models have been proposed which can be broadly classified in the following categories.

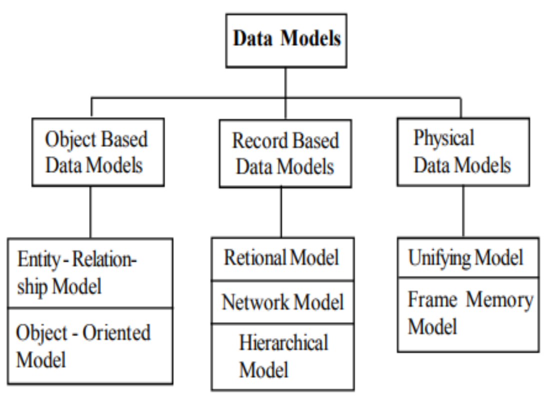


Figure 2: - Data Modeling

Classification of Data Models

1. Object based data models
2. Record – based data models
3. Physical data models

Object-based logical models

Object-based logical models are used in describing data at logical and view levels. They are characterized by the fact they provide flexible structuring capabilities and allow data constraints to be specified explicitly. There are many different data models, some of them are

1. The Entity-relationship model.
2. The Object-oriented model.
3. The semantic data model.
4. The Functional data model.

Record-based data models

These models are used to specify the overall logical structure of the database. With some models a higher level description of the implementation of the structure of the database can also be specified explicitly. The data integrity constraints cannot be specified explicitly with these models. In record based data models, the database is structured in fixed formats records of several types. Each record defines fixed number of fields (attributes) and each field is fixed length. These models are used to specify the overall logical structure of the database and are used in describing the database at conceptual level. The three widely accepted record – based data models are

1. Relational model
2. Network model
3. Hierarchical model

Physical Data Models

Physical data model are used to describe data at the lowest level. In contrast to logical data models, there are few number of physical data models which are in use. Very few physical data models have been proposed so far.

Two of these well known models are the unifying model and the frame memory model.

--

1.4. ENTITIES AND THEIR ATTRIBUTES

Entity-relationship model is a model used for design and representation of relationships between data.

The main data objects are termed as Entities, with their details defined as attributes, some of these attributes are important and are used to identity the entity, and different entities are related using relationships.

1.4.1. Entities

They are represented using the rectangle shape box. These rectangles are named with the entity set they represent.

ER modeling is a top-down structure to database design that begins with identifying the important data called entities and relationships in combination with the data that must be characterized in the model. Then database model designers can add more details such as the information they want to hold about the entities and relationships which are the attributes and any constraints on the entities, relationships, and attributes. ER modeling is an important technique for any database designer to master and forms the basis of the methodology.

- **Entity type:** It is a group of objects with the same properties that are identified by the enterprise as having an independent existence. An entity type has an independent existence within a database.
- **Entity occurrence:** A uniquely identifiable object of an entity type.

1.4.2. Attributes

Attributes are the properties of entities that are represented using ellipse shaped figures. Every elliptical figure represents one attribute and is directly connected to its entity (which is represented as a rectangle).

1.4.3. Relationship

A relationship type is a set of associations between one or more participating entity types. Each relationship type is given a name that describes its function.

The entities occupied in a particular relationship type are referred to as participants in that relationship. The number of participants involved in a relationship type is termed as the degree of that relationship.

A diamond-shaped box represents relationships. All the entities (rectangle shaped) participating in a relationship gets connected using a line.

There are four types of relationships. These are:

- One-to-one: When only a single instance of an entity is associated with the relationship, it is termed as '1:1'.
- One-to-many: When more than one instance of an entity is related and linked with a relationship, it is termed as '1:N'.
- Many-to-one: When more than one instance of an entity is linked with the relationship, it is termed as 'N:1'.
- Many-to-many: When more than one instance of an entity on the left and more than one instance of an entity on the right can be linked with the relationship, then it is termed as N:N relationship.

1.5. ADVANTAGES AND DISADVANTAGES OF DBMS

A database management system (DBMS) refers to the technology for creating and managing databases. DBMS is a software tool to organize (create, retrieve, update and manage) data in a database.

The main aim of a DBMS is to supply a way to store up and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have embedded meaning.

A DBMS manage data and has many advantages. These are:

- Data independence: Application programs should be as free or independent as possible from details of data representation and storage. DBMS can supply an abstract view of the data for insulating application code from such facts.
- Efficient data access: DBMS utilizes a mixture of sophisticated concepts and techniques for storing and retrieving data competently, and this feature becomes important in cases where the data is stored on external storage devices.
- Data integrity and security: If data is accessed through the DBMS, the DBMS can enforce integrity constraints on the data.
- Data administration: When several users share the data, integrating the administration of data can offer major improvements. Experienced professionals understand the nature of the data being managed and can be responsible for organizing the data representation to reduce redundancy and make the data to retrieve efficiently.

Disadvantages of DBMS

- It's Complexity
- Except MySQL, which is open source, licensed DBMSs are generally costly.
- They are large in size.

1.6. DBMS VS RDBMS

A DBMS is software used to store and manage data. The DBMS was introduced during 1960's to store any data. It also offers manipulation of the data like insertion, deletion, and updating of the data.

DBMS system also performs the functions like defining, creating, revising and controlling the database. It is specially designed to create and maintain data and enable the individual business application to extract the desired data.

Relational Database Management System (RDBMS) is an advanced version of a DBMS system. It came into existence during 1970's. RDBMS system also allows the organization to access data more efficiently then DBMS.

RDBMS is a software system which is used to store only data which need to be stored in the form of tables. In this kind of system, data is managed and stored in rows and columns which is known as tuple and attributes. RDBMS is a powerful data management system and is widely used across the world.

The main differences between DBMS and RDBMS are given below:

No.	DBMS	RDBMS
1)	DBMS applications store data as file.	RDBMS applications store data in a tabular form.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	Normalization is not present in DBMS.	Normalization is present in RDBMS.
4)	DBMS does not apply any security with regards to data manipulation.	RDBMS defines the integrity constraint for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be no relation between the tables.	in RDBMS, data values are stored in the form of tables, so a relationship between these data values will be stored in the form of a table as well.
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS does not support distributed database.	RDBMS supports distributed database.
8)	DBMS is meant to be for small organization and deal with small data. it supports single user.	RDBMS is designed to handle large amount of data. it supports multiple users.
9)	Examples of DBMS are file systems, xml etc.	Example of RDBMS are mysql, postgre, sql server, oracle etc.

1.7. LET US SUM UP

In this unit, you have learnt about the database & database systems, significance and applications of database system, entities and attributes of database system, and advantages and disadvantages of database. This knowledge would make you understand the basics of database and database management system concepts and comparison of DBMS with RDBMS used to solve the real time problems. Thus, the database management system concepts unit would have brought you to closer to know the concept of database systems.

1.8. UNIT – END QUESTIONS

1. List out the basic concepts of database management system.
2. Explain about the relationship between categories of programming languages.
3. Write about the advantages and disadvantages of DBMS.

1.9. ANSWER TO CHECK YOUR PROGRESS

1. A Database is a collection of related data organised in a way that data can be easily accessed, managed and updated. Database can be software based or hardware based, with one sole purpose, storing data. A DBMS is software that allows creation, definition and manipulation of database, allowing users to store, process and analyse data easily. DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more. DBMS also provides protection and security to the databases. It also maintains data consistency in case of multiple users. Database management systems are important to businesses and organizations because they provide a highly efficient method for handling multiple types of data. Some of the data that are easily managed with this type of system include: employee records, student information, payroll, accounting, project management, inventory and library books. These systems are built to be extremely versatile.

2. ER modeling is a top-down structure to database design that begins with identifying the important data called entities and relationships in combination with the data that must be characterized in the model. A relationship type is a set of associations between one or more participating entity types. Each relationship type is given a name that describes its function.

Attributes are the properties of entities that are represented using ellipse shaped figures. Every elliptical figure represents one attribute and is directly connected to its entity.

3. The main aim of a DBMS is to supply a way to store up and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have embedded meaning. Disadvantages of DBMS are, it's Complexity, except MySQL, which is open source, licensed DBMSs are generally costly and they are large in size.

.

UNIT II – DATABASE SYSTEM ARCHITECTURE

Structure

2.1. INTRODUCTION

In this lesson you will be aware with the basic elements used to construct and manage the databases. These elements include the architecture, facilities, DDL, DML, structure of DBMS and players of DBMS. These basic elements are used to construct and manage the more comprehensive database components. Some of the basic elements needs very detailed

information, however, the purpose of this type of basic elements is to introduce certain basic concepts and to provide some necessary definitions for the topics that follow in next few lessons.

2.2. OBJECTIVES

After going through this lesson you would be in a positions to

- Three levels of architecture.
- DDL & DML languages.
- Define data dictionary and distributed processing.
- Explain the concept of ICT.

2.3. THREE LEVEL ARCHITECTURE OF DBMS

DBMS architecture helps in design, development, implementation, and maintenance of a database. A database stores critical information for a business. Selecting the correct Database Architecture helps in quick and secure access to this data.

There are mainly three levels of data abstraction:

1. Internal Level: Actual PHYSICAL storage structure and access paths.
2. Conceptual or Logical Level: Structure and constraints for the entire database.
3. External or View level: Describes various user views.

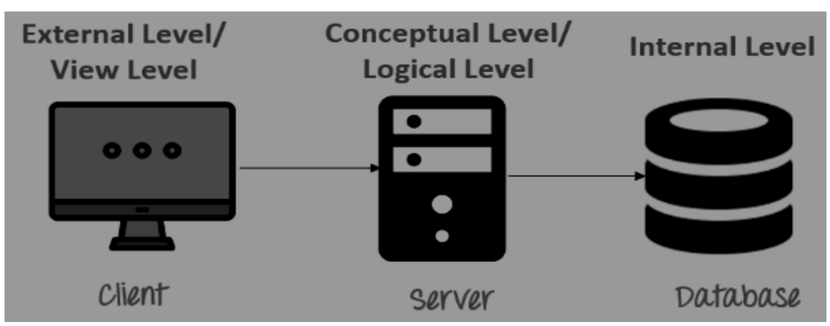


Figure 3: - Three Level Architecture of DBMS

2.3.1. External Schema/Level

An external schema describes the part of the database which specific user is interested in. It hides the unrelated details of the database from the user. There may be "n" number of external views for each database.

Each external view is defined using an external schema, which consists of definitions of various types of external record of that specific view.

An external view is just the content of the database as it is seen by some specific particular user. For example, a user from the sales department will see only sales related data.

Facts about external schema:

- An external level is only related to the data which is viewed by specific end users.
- This level includes some external schemas.
- External schema level is nearest to the user.
- The external schema describes the segment of the database which is needed for a certain user group and hides the remaining details from the database from the specific user group.

2.3.2. Conceptual Schema/Level

The conceptual schema describes the Database structure of the whole database for the community of users. This schema hides information about the physical storage structures and focuses on describing data types, entities, relationships, etc.

This logical level comes between the user level and physical storage view. However, there is only single conceptual view of a single database.

Facts about Conceptual schema:

- Defines all database entities, their attributes, and their relationships.
- Security and integrity information.
- In the conceptual level, the data available to a user must be contained in or derivable from the physical level.

2.3.3. Internal Level/Schema

The internal schema defines the physical storage structure of the database. The internal schema is a very low-level representation of the entire database. It contains multiple occurrences of multiple types of internal record. In the ANSI term, it is also called "stored record'.

Facts about Internal schema:

- The internal schema is the lowest level of data abstraction.
- It helps you to keeps information about the actual representation of the entire database. Like the actual storage of the data on the disk in the form of records.
- The internal view tells us what data is stored in the database and how.
- It never deals with the physical devices. Instead, internal schema views a physical device as a collection of physical pages.

DBMS Architecture allows you to make changes on the presentation level without affecting the other two layers

----

2.4. MAPPING

The DBMS is responsible for mapping between the three types of schemas. It must be capable of checking the schema for consistency and must use the info. In the schema to map between external schema and internal schema via the conceptual schema as shown in the figure below:-

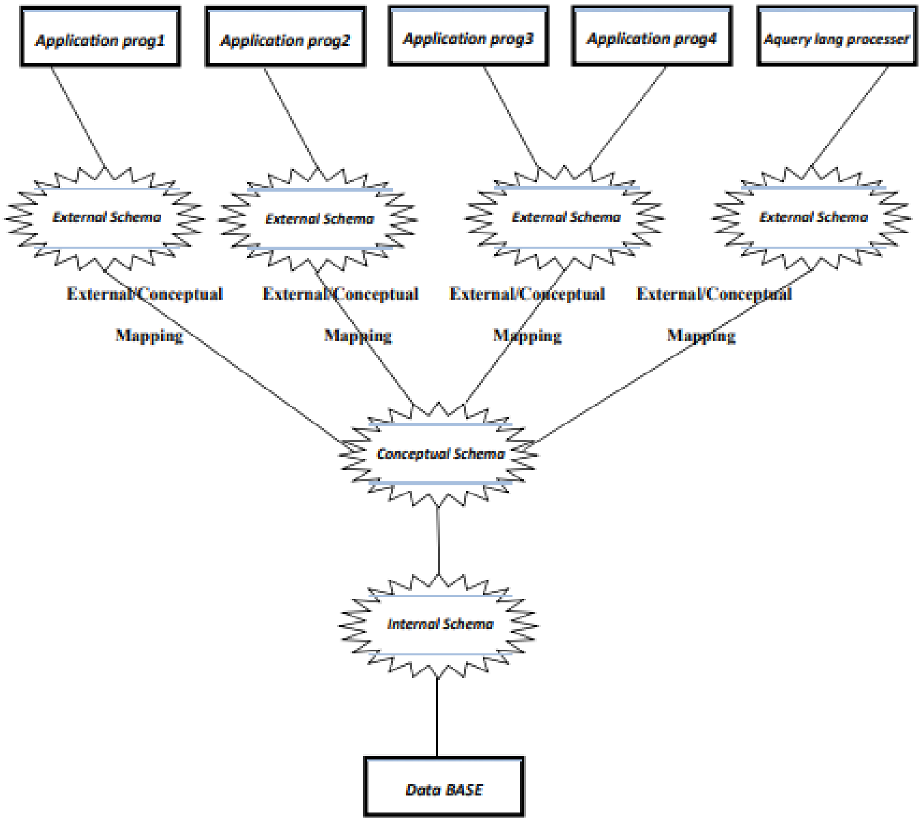


Figure 4: - Mapping of DBMS Architecture

Process of transforming request and results between three level it's called mapping. There are the two types of mappings:

1. Conceptual/Internal Mapping
2. External/Conceptual Mapping

1. Conceptual/Internal Mapping:

- The conceptual/internal mapping defines the correspondence between the conceptual view and the store database.
- It specifies how conceptual record and fields are represented at the internal level.
- It relates conceptual schema with internal schema.
- If structure of the store database is changed.
- If changed is made to the storage structure definition-then the conceptual/internal mapping must be changed accordingly, so that the conceptual schema can remain invariant.
- There could be one mapping between conceptual and internal levels.

2. External/Conceptual Mapping:

- The external/conceptual mapping defines the correspondence between a particular external view and conceptual view.
- It relates each external schema with conceptual schema.
- The differences that can exist between these two levels are analogous to those that can exist between the conceptual view and the stored database.
- Example: fields can have different data types; fields and record name can be changed; several conceptual fields can be combined into a single external field.
- Any number of external views can exist at the same time; any number of users can share a given external view: different external views can overlap.
- There could be several mapping between external and conceptual levels.

2.5. MySQL ARCHITECTURE

MySQL is very different from other database servers, and its architectural characteristics make it useful for a wide range of purposes as well as making it a poor choice for others. MySQL is not perfect, but it is flexible enough to work well in very demanding environments, such as web applications. At the same time, MySQL can power embedded applications, data warehouses, content indexing and delivery software, highly available redundant systems, online transaction processing (OLTP), and much more.

To get the most from MySQL, you need to understand its design so that you can work with it, not against it. MySQL is flexible in many ways. For example, you can configure it to run well on a wide range of hardware, and it supports a variety of data types. However, MySQL's most unusual and important feature is its storage-engine architecture, whose design separates query processing and other server tasks from data storage and retrieval.

MySQL architecture is broken into three layers basically which can be defined by,

1. Application Layer.
2. Logical Layer.
3. Physical Layer.

1. Application Layer

The application layer is where the clients and users interact with the MySQL RDBMS. All the services needed for connection handling, authentication, security are here. There are three main components in this layer namely Administrators, Clients, Query Users as shown in the below figure.

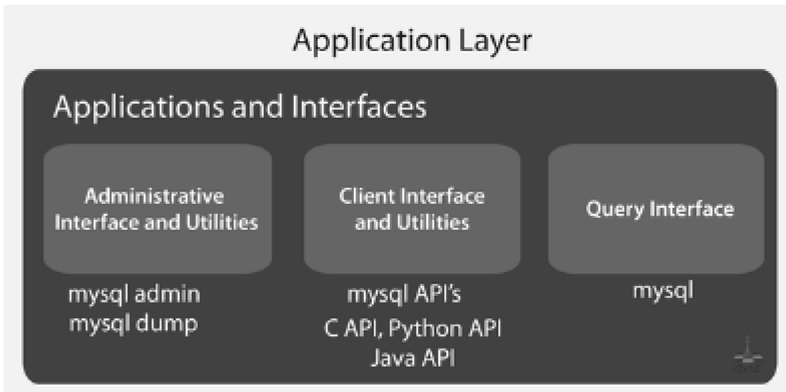


Figure 5: - Application Layer

- Administrators

  Administrators use various administrative interface and utilities like mysqladmin which performs tasks like shutting down the server and creating or dropping databases, mysqldump for backing up the database or copying databases to another server.
- Clients

  Clients communicate with MySQL through various interfaces and utilities like MySQL API's. The MySQL API sends the query to the server as a series of tokens.
- Query User

  The query users interact with MySQL RDBMS through a query interface that is *mysql*.

2. Logical Layer

The Logical Layer takes the data from the Application Layer. Any functionality provided across storage engines lives at this level like stored procedures, triggers and views. It is divided into subsystems like Query Processor, Transaction Management, Recovery Management, and Storage Management. These subsystems work together to process the requests issued to the MySQL database server. The output of one of the above subsystems becomes the input for another. Below is the basic conceptional diagram.

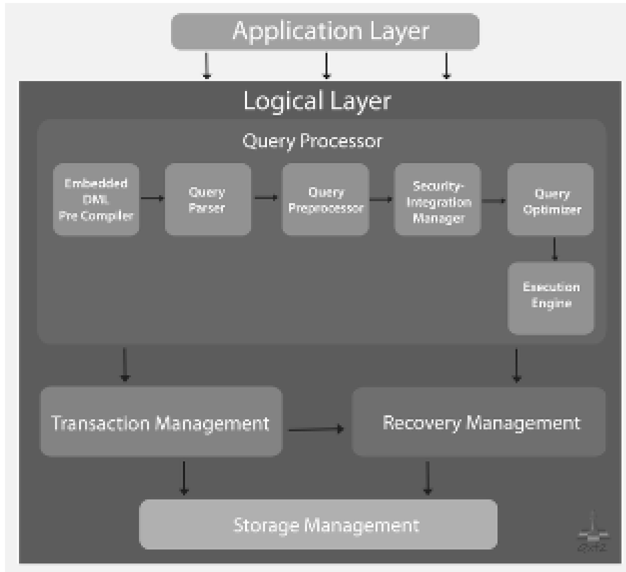


Figure 6: - Logical Layer

3. Physical Layer

The third layer is the Physical Layer which contains the storage engines. They are responsible for storing and retrieving all data stored in MySQL. Physical Layer of MySQL is slightly different from other RDBMS. Here the physical system consists of Pluggable Storage Engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

MySQL stores each database (also called a schema) as a subdirectory of its data directory in the underlying file system. Every database has a corresponding data directory. When you create a table, MySQL stores the table definition in a .frm file with the same name as the table.

2.6. SQL SERVER 2000 ARCHITECTURE

Microsoft SQL Server provides a robust, if simpler, configuration for enterprise database applications. Microsoft SQL Server uses a set of kernel-based libraries to manage the internal database operations. Microsoft SQL Server divides its mechanism for data storage into the concept of physical and logical storage into the physical medium of file groups, files, transaction logs and the logical aspect into databases. Each database within a SQL Server environment is assigned to its own set of file group and transaction logs stored on disk. Unlike an Oracle

environment that has a one-to-one correlation of one instance per database, the Microsoft SQL Server 2000 architecture is quite a bit different.

SQL Server 2000 provides a set of components that work together to meet the needs of the largest data processing systems and commercial Web sites while providing easy-to-use data storage services to an individual or small business.

SQL Server 2000 consists of numerous components that interact to provide complete database application capabilities, including relational database management, OLAP, data mining, full-text indexing, data import and export, and replication, as well as client access, as depicted in Figure 4.1. In the later chapters of this book, we review each of these components in detail and assist you in configuring and using them in your applications. This chapter begins by exploring the base components of SQL Server and its databases.

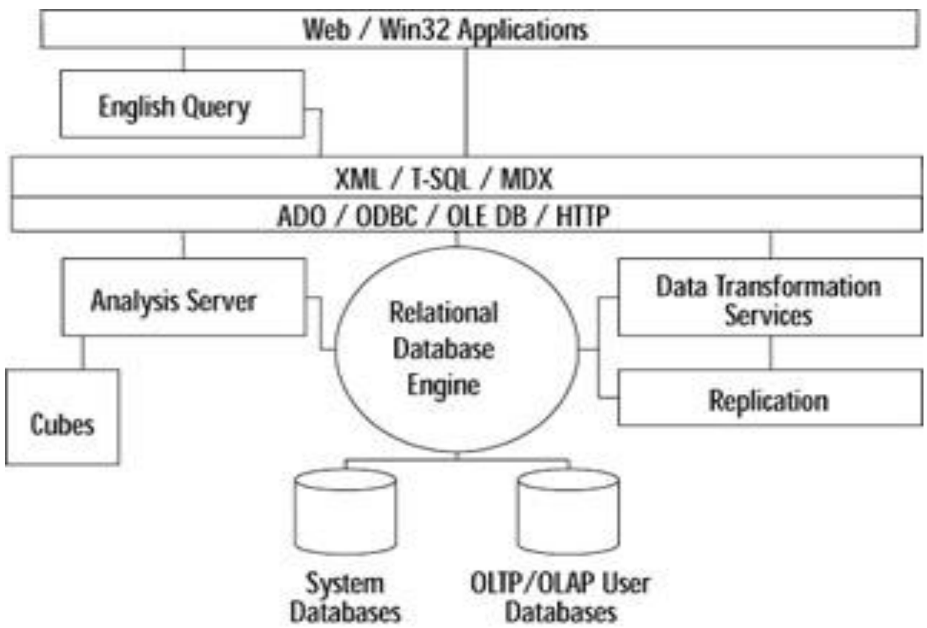


Figure 7: - SQL Server 2000 Architecture

SQL Server 2000 data is stored in databases. The data in a database is organized into the logical components that are visible to users, while the database itself is physically implemented as two or more files on disk.

Logical Database Components

The logical database components include objects, collations, logins, users, roles, and groups.

An object in a SQL Server 2000 database can use a collation different from another object within that same database.

Collations control the physical storage of character strings in SQL Server 2000. A collation specifies the bit patterns that represent each character and the rules by which characters are sorted and compared.

Logins, users, roles, and groups are the foundation for the security mechanisms of SQL Server 2000. Users who connect to SQL Server must identify themselves by using a Specific Login Identifier (ID). Users can then see only the tables and views that they are authorized to see and can execute only the stored procedures and administrative functions that they are authorized to execute. This system of security is based on the IDs used to identify users.

Physical Database Architecture

The fundamental unit of data storage in SQL Server is the page. In SQL Server 2000, the page size is 8 kilobytes (KB). In other words, SQL Server 2000 databases contain 128 pages per megabyte (MB).

The start of each page is a 96-byte header used to store system information, such as the type of page, the amount of free space on the page, and the object ID of the object owning the page.

Extents are the basic unit in which space is allocated to tables and indexes. An extent is eight contiguous pages, or 64 KB. In other words, SQL Server 2000 databases have 16 extents per megabyte.

2.7. ORACLE ARCHITECTURE

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.

The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

In the following sections we discuss the main components of the Oracle DBMS architecture and the logical and physical database structures.

Logical Database Structures

For the architecture of an Oracle database we distinguish between logical and physical database structures that make up a database. Logical structures describe logical areas of storage (name spaces) where objects such as tables can be stored. Physical structures, in contrast, are determined by the operating system files that constitute the database. The logical database structures include:

- **Database: -**

  A database consists of one or more storage divisions, so-called tablespaces.
- **Tablespaces: -**

  A tablespace is a logical division of a database. All database objects are logically stored in tablespaces.
- **Segments: -**

  If a database object (e.g., a table or a cluster) is created, automatically a portion of the tablespace is allocated. This portion is called a segment..
- **Extent: -**

  An extent is the smallest logical storage unit that can be allocated for a database object, and it consists a contiguous sequence of data blocks.

Physical Database Structure

The physical database structure of an Oracle database is determined by files and data blocks:

- **Data Files: -**

  A tablespace consists of one or more operating system files that are stored on disk. Thus a database essentially is a collection of data files that can be stored on different storage devices.
- **Blocks: -**

  An extent consists of one or more contiguous Oracle data blocks. A block determines the finest level of granularity of where data can be stored. One data block corresponds to a specific number of bytes of physical database space on disk.
- **Redo-Log Files: -**

  Each database instance maintains a set of redo-log files. These files are used to record logs of all transactions. The logs are used to recover the database's transactions in their proper order in the event of a database crash.
- **Control Files: -**

  Each database instance has at least one control file. In this file the name of the database instance and the locations (disks) of the data files and redo-log files are recorded.
- **Archive/Backup Files: -**

If an instance is running in the archive-log mode, the ARCH process archives the modifications of the redo-log files in extra archive or backup files. In contrast to redo-log files, these files are typically not overwritten.

2.8. DATABASE MANAGEMENT SYSTEM FACILITIES

Typically, a DBMS provides the following facilities

- **Data Definition Language (DDL)**

  It allows a database designer to define the database using a Data Definition Language (DDL) provided for the particular DBMS. The DDL allows the designer to specify the data types and structures, and the constraints on the data to be stored in the database.

- **Data Manipulation Language (DML)**

  It allows users to insert, update, delete and retrieve data from the database through a Data Manipulation Language (DML). Having a central repository for all data and data descriptions allows the DML to provide a general enquiry facility to this data, called a query language. Using a query language, directly or indirectly, enables new lines of enquiry to be constructed and satisfied quickly. A query language is sufficiently high level to allow non-technical personnel to use it, easily. The most common query language is the Structured Query Language (SQL –pronounced 'S-Q-L').

- **View Mechanism**

  The DBMS provides a view mechanism that allows each user to have his or her own view of the database. The DDL is used to define a view that is a subset of the database.

- **Multiple indexes**

  An index is a mechanism for reducing the time taken to find a specific item of data in a database. A database index works in a similar way to a user of the index in this book. If you want to use this book to find out about the topic "multiple indexes"then you have a choice.

  An index in a database can store each value of an indexed data item (field), e.g. student enrolment number, together with the page number in the storage medium where the data belonging to this value is stored. For example, information stored in a database about a particular student such as surname, home address, et cetera may be quickly found if an index has been created that stores every student's enrolment number and the location of the corresponding information.

Indexes may also be created on other fields of a student's record, e.g. surname, address. However, since some fields such as the surname are unlikely to be unique, the entries in the index may reference more than one location, just like the entries in the index for this book. An index on a unique field is known as a primary index whereas an index on a non-unique field is known as a secondary index.

- **Indexing Overheads**

  Indexes have to be constantly kept up to date. When a new data value is added or modified the corresponding index must be updated. This takes time. This is called an update overhead.

2.9. DATA DEFINITION LANGUAGE

Data Definition language (DDL) in DBMS with Examples: Data Definition Language can be defined as a standard for commands through which data structures are defined. It is a computer language that used for creating and modifying the structure of the database objects, such as schemas, tables, views, indexes, etc. Additionally, it assists in storing the metadata details in the database.

Some of the common Data Definition Language commands are:

- CREATE
- ALTER
- DROP

CREATE- Data Definition language (DDL)

The main use of the create command is to build a new table and it comes with a predefined syntax. It creates a component in a relational database management system. There are many implementations that extend the syntax of the command to create the additional elements, like user profiles and indexes.

The general syntax for the create command in Data Definition Language is mentioned below:

CREATE TABLE tablename (Column1 DATATYPE, Column2 DATATYPE, Column3 DATATYPE, …….. ColumnN DATATYPE)

ALTER- Data Definition language (DDL)

An existing database object can be modified by the ALTER statement. Using this command, the users can add up some additional column and drop existing columns. Additionally, the data type of columns involved in a database table can be changed by the ALTER command.

The general syntax of the ALTER command is mentioned below:

ALTER TABLE table_name ADD column_name (for adding a new column).
ALTER TABLE table_name RENAME To new_table_name (for renaming a table).
ALTER TABLE table_name MODIFY column_name data type (for modifying a column).
ALTER TABLE table_name DROP COLUMN column_name (for deleting a column).

Drop- Data Definition language (DDL)

By the use of this command, the users can delete an index, table or view. A component from a relational database management system can be removed by a DROP statement in SQL.

There are many systems that allow the DROP and some other Data Definition Language commands for occurring inside a transaction and then it can be rolled back.

The object will not be available for use once the DROP statement executed.

The General syntax of the Drop command is mentioned below:

DROP TABLE table_name;
DROP DATABASE database_name;
DROP TABLE Student;
DROP TABLE index_name;

Truncate- Data Definition language (DDL)

By using the Truncate command, the users can remove the table content, but the structure of the table is kept. In simple language, it removes all the records from the table structure. The users can't remove data partially through this command. In addition to this, every space allocated for the data is removed by Truncate command.

The syntax of the Truncate command is mentioned below:

TRUNCATE TABLE table_name;
TRUNCATE TABLE Student;

2.10. DATA MANIPULATION LANGUAGE

Data Manipulation Language (DML) can be defined as a set of syntax elements that are used to manage the data in the database. The commands of DML are not auto-committed and modification made by them is not permanent to the database. It is a computer programming language that is used to perform select, insert, delete and update data in a database. The user requests are assisted by Data Manipulation Language. This language is responsible for all forms of data modification in a database.

A data manipulation language (DML) is a family of computer languages including commands permitting users to manipulate data in a database. This manipulation involves inserting data into database tables, retrieving existing data, deleting data from existing tables and modifying existing data. DML is mostly incorporated in SQL databases.

DML resembles simple English language and enhances efficient user interaction with the system. The functional capability of DML is organized in manipulation commands like SELECT, UPDATE, INSERT INTO and DELETE FROM, as described below:

- **SELECT:** This command is used to retrieve rows from a table. The syntax is SELECT [column name(s)] from [table name] where [conditions]. SELECT is the most widely used DML command in SQL.
- **UPDATE:** This command modifies data of one or more records. An update command syntax is UPDATE [table name] SET [column name = value] where [condition].
- **INSERT:** This command adds one or more records to a database table. The insert command syntax is INSERT INTO [table name] [column(s)] VALUES [value(s)].
- **DELETE:** This command removes one or more records from a table according to specified conditions. Delete command syntax is DELETE FROM [table name] where [condition].

2.11. DATABASE MANAGEMENT SYSTEM STRUCTURE

A database is an organized collection of data. Instead of having all the data in a list with a random order, a database provides a structure to organize the data. One of the most common data structures is a database table. A database table consists of rows and columns. A database table is also called a two-dimensional array. An array is like a list of values, and each value is identified by a specific index. A two-dimensional array uses two indices, which correspond to the rows and columns of a table.

In database terminology, each row is called a record. A record is also called an object or an entity. In other words, a database table is a collection of records. The records in a table are the objects you are interested in, such as the books in a library catalog or the customers in a sales database. A field corresponds to a column in the table and represents a single value for each record. A field is also called an attribute. In other words, a record is a collection of related attributes that make up a single database entry.

DBMS (Database Management System) acts as an interface between the user and the database. The user requests the DBMS to perform various operations (insert, delete, update and retrieval) on the database. The components of DBMS perform these requested operations on the database and provide necessary data to the users. The various components of DBMS are shown below: -

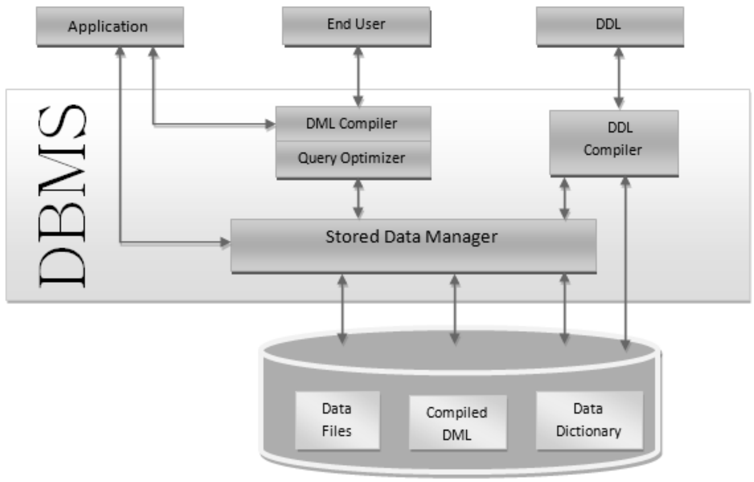


Figure 8: - DBMS Structure

1. DDL Compiler - Data Description Language compiler processes schema definitions specified in the DDL. It includes metadata information such as the name of the files, data items, storage details of each file, mapping information and constraints etc.

2. DML Compiler and Query optimizer - The DML commands such as insert, update, delete, retrieve from the application program are sent to the DML compiler for compilation into object code for database access. The object code is then optimized in the best way to execute a query by the query optimizer and then send to the data manager.

3. Data Manager - The Data Manager is the central software component of the DBMS also knows as Database Control System.

The Main Functions Of Data Manager Are: –

- Convert operations in user's Queries coming from the application programs or combination of DML Compiler and Query optimizer which is known as Query Processor from user's logical view to physical file system.
- Controls DBMS information access that is stored on disk.
- It also controls handling buffers in main memory.
- It also enforces constraints to maintain consistency and integrity of the data.
- It also synchronizes the simultaneous operations performed by the concurrent users.
- It also controls the backup and recovery operations.

4. Data Dictionary - Data Dictionary is a repository of description of data in the database. It contains information about

- Data - names of the tables, names of attributes of each table, length of attributes, and number of rows in each table.
- Relationships between database transactions and data items referenced by them which is useful in determining which transactions are affected when certain data definitions are changed.
- Constraints on data i.e. range of values permitted.
- Detailed information on physical database design such as storage structure, access paths, files and record sizes.
- Access Authorization - is the Description of database users their responsibilities and their access rights.
- Usage statistics such as frequency of query and transactions.

Data dictionary is used to actually control the data integrity, database operation and accuracy. It may be used as a important part of the DBMS.

Importance of Data Dictionary -

Data Dictionary is necessary in the databases due to following reasons:

- It improves the control of DBA over the information system and user's understanding of use of the system.
- It helps in documentating the database design process by storing documentation of the result of every design phase and design decisions.
- It helps in searching the views on the database definitions of those views.
- It provides great assistance in producing a report of which data elements (i.e. data values) are used in all the programs.
- It promotes data independence i.e. by addition or modifications of structures in the database application program are not affected.

5. Data Files - It contains the data portion of the database.

6. Compiled DML - The DML complier converts the high level Queries into low level file access commands known as compiled DML.

7. End Users - They are already discussed in previous section.

2.12. DATABASE MANAGER

The data manager is the central software component of the DBMS. It is sometimes referred to as the database control system. One of the functions of the data manager is to convert operations in the user’s queries coming directly via the query processor or indirectly via an application program from user’s logical view to a physical file system. The data manager is responsible for interfacing with the file system. In addition, the tasks of enforcing constraints to

maintain the consistency and integrity of the data, as well as its security, are also performed by the data manager. Synchronizing the simultaneous operations performed by concurrent users is under the control of the data manager. It is also entrusted with the backup and recovery operations.

A database manager (DB manager) is a computer program, or a set of computer programs, that provide basic database management functionalities including creation and maintenance of databases. Database managers have several capabilities including the ability to back up and restore, attach and detach, create, clone, delete and rename the databases.

Database managers are used to manage local and remote databases. They discover databases based on the Web server and provide the ability to connect to any of the databases residing in the network. They provide a handful of administrative functionalities such as managing tables, views and stored procedures, as well as run ad hoc queries.

DB managers connect to the database and display information from catalogs that are part of a database. DB managers can have a set of command-line parameters, which allow them to initiate features and functions external to the graphical user interface.

DB managers allow database administrators to define new patches for databases or to easily apply new patches that come from vendors, thus updating databases with enhancements and keeping them secure.

2.13. DATABASE ADMINISTRATOR

Centralized control of the database is exerted by a person or group of persons under the supervision of a high-level administrator. This person or group is referred to as the database administrator (DBA). They are the users who are most familiar with the database and are responsible for creating, modifying, and maintaining its three levels. Database Administrator is responsible to manage the DBMS's use and ensure that the database is functioning properly.

DBA administers the three levels of database and consultation with the overall user community, sets up the definition of the global view of the various users and applications and is responsible the definition and implementation of the internal level, including the storage structure and access methods to be used for the optimum performance of the DBMS. DBA is responsible for granting permission to the users of the database and stores the profile of each user in the database.

Responsibilities

Designing & creating relational database objects such as tables, views & indexes; Supporting and maintaining the environment a relational database requires to properly function (i.e., security, recovery, backup & reorganizations) ; Ensuring that relational database access code performs efficiently (i.e., SQL review, database monitoring).

Activities - Central DBA

- Developing & maintaining naming standards for database objects such as tablespaces, tables, indexes & views
- Participating in database migration reviews
- Assisting in product installation & reviewing initial installation options; for Oracle, the Central DBA will assist in product installation and specify initial installation options
- Providing functional guidance to the systems programmer & the operator. For Oracle, the Central DBA will provide this functional guidance to the Unix System Administrator.
- Evaluating & testing DBMS related software
- Develop operational procedures • Supporting, monitoring & tuning the database subsystems & instances; for Oracle this includes starting and stopping the instances, listeners, and intelligent agents
- Participate in selecting database management support tools
- Developing & implementing database administration policies & procedures including subsystem or instance security guidelines

These are the functions of a data administrator (not to be confused with database administrator functions):

- Data policies, procedures, standards
- Planning- development of organization's IT strategy, enterprise model, cost/benefit model, design of database environment, and administration plan.
- Data conflict (ownership) resolution
- Data analysis- Define and model data requirements, business rules, operational requirements, and maintain corporate data dictionary
- Internal marketing of DA concepts
- Managing the data repository

2.14. DATA DICTIONARY

A data dictionary contains metadata i.e data about the database. The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc. The users of the database normally don't interact with the data dictionary; it is only handled by the database administrators.

The data dictionary in general contains information about the following:

- Names of all the database tables and their schemas.
- Details about all the tables in the database, such as their owners, their security constraints, when they were created etc.
- Physical information about the tables such as where they are stored and how.

- Table constraints such as primary key attributes, foreign key information etc.
- Information about the database views that are visible.
- The different types of data dictionary are:
- Active Data Dictionary
- If the structure of the database or its specifications changes at any point of time, it should be reflected in the data dictionary. This is the responsibility of the database management system in which the data dictionary resides.
- So, the data dictionary is automatically updated by the database management system when any changes are made in the database. This is known as an active data dictionary as it is self updating.
- Passive Data Dictionary
- This is not as useful or easy to handle as an active data dictionary. A passive data dictionary is maintained separately to the database whose contents are stored in the dictionary. That means that if the database is modified the database dictionary is not automatically updated as in the case of Active Data Dictionary.
- So, the passive data dictionary has to be manually updated to match the database. This needs careful handling or else the database and data dictionary are out of sync.

A data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects, and other data.

The data dictionary is a crucial component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

In a relational database, the metadata in the data dictionary includes the following:

- Names of all tables in the database and their owners
- Names of all indexes and the columns to which the tables in those indexes relate
- Constraints defined on tables, including primary keys, foreign-key relationships to other tables, and not-null constraints

For most relational database management systems (RDBMS), the database management system software needs the data dictionary to access the data within a database. For example, the Oracle DB software has to read and write to an Oracle DB. However, it can only do this via the data dictionary created for that particular database.

2.15. DISTRIBUTED PROCESSING

In distributed processing, a database's logical processing is shared among two or more physically independent sites that are connected through a network. For example, the data

input/output (I/O), data selection, and data validation might be performed on one computer, and a report based on that data might be created on another computer.

A distributed database, on the other hand, stores a logically related database over two or more physically independent sites. The sites are connected via a computer network. In contrast, the distributed processing system uses only a single-site database but shares the processing chores among several sites. In a distributed database system, a database is composed of several parts known as database fragments. The database fragments are located at different sites and can be replicated among various sites. Each database fragment is, in turn, managed by its local database process.

- Distributed processing does not require a distributed database, but a distributed database requires distributed processing.
- Distributed processing may be based on a single database located on a single computer. For the management of distributed data to occur, copies or parts of the database processing functions must be distributed to all data storage sites.
- Both distributed processing and distributed databases require a network to connect all components.

Distributed data processing is a computer-networking method in which multiple computers across different locations share computer-processing capability. This is in contrast to a single, centralized server managing and providing processing capability to all connected systems. Computers that comprise the distributed data-processing network are located at different locations but interconnected by means of wireless or satellite links.

Lower Cost

Larger organizations invest in expensive mainframe and supercomputers to function as centralized servers. Each mainframe machine,

Reliable

Hardware glitches and software anomalies can cause single-server processing to malfunction and fail, resulting in a complete system breakdown. Distributed data processing is more reliable, since multiple control centers are spread across different machines.

Improved Performance and Reduced Processing Time

Single computers are limited in their performance and efficiency. An easy way to increase performance is by adding another computer to a network. Adding yet another computer will further augment performance, and so on. Distributed data processing works on this principle and holds that a job gets done faster if multiple machines are handling it in parallel, or synchronously.

Flexible

Individual computers that comprise a distributed network are present at different geographical locations. For example, an organizational-distributed network comprising of three computers can have each machine in a different branch. The three machines are interconnected via the Internet and are able to process data in parallel, even while at different locations. This

makes distributed data-processing networks more flexible. The system is flexible also in terms of increasing or decreasing processing power.

2.16. INFORMATION AND COMMUNICATION TECHNOLOGY SYSTEM

Information and communications technology (ICT) is an extensional term for information technology (IT) that stresses the role of unified communications and the integration of telecommunications (telephone lines and wireless signals) and computers, as well as necessary enterprise software, middleware, storage, and audiovisual systems, that enable users to access, store, transmit, and manipulate information.

ICT is technology that supports activities involving information. Such activities include gathering, processing, storing and presenting data. Increasingly these activities also involve collaboration and communication. Hence IT has become ICT: information and communication technology.

The term ICT is also used to refer to the convergence of audiovisual and telephone networks with computer networks through a single cabling or link system. There are large economic incentives to merge the telephone network with the computer network system using a single unified system of cabling, signal distribution, and management. ICT is an umbrella term that includes any communication device, encompassing radio, television, cell phones, computer and network hardware, satellite systems and so on, as well as the various services and appliance with them such as video conferencing and distance learning.

ICT is a broad subject and the concepts are evolving. It covers any product that will store, retrieve, manipulate, transmit, or receive information electronically in a digital form (e.g., personal computers, digital television, email, or robots). The different types of communication in ICT include electronic mail, video conferencing, facsimile and telephone conferencing. ICT communication deals with storage, retrieval transmission and manipulation of digital information. ICT communication uses ICT devices to connect businesses, organizations and individuals.

Electronic mail is the common form of electronic communication used for transmitting and receiving digital information. Emails are essential in sending messages, pictures files and other attachments. Firms and organizations use emails for business purposes and as a medium for communication with employees, personnel and clients. Facsimile is another common means of ICT communication used for sending messages over the telephone network. Modern fax machines are digital, making it possible to send a message over a wireless connection. Faxes can be sent over a wireless connection and received by the fax machine of the recipient.

Video conferencing is the best communication medium when companies want to reach different people across different time zones or countries. This medium uses a camera, loudspeakers, Internet connections and microphone to connect different people at the same time. The equipment used allows everyone to see, speak and listen to each other. Another type of ICT communication is telephone conferencing. Phone conferences allow participants to listen to each other. They are connected through a phone call using an option of conferencing. Participants enter a unique code or number to bridge the call.

Useful concept of ICT

It depends on the local culture and the particular ICT available and how it is configured and managed. The understanding, management and configuration of the available technology might vary the concept of ICT from

- A collection of tools and devices used for particular tasks, eg, publishing, course delivery, and transaction processing...
- An organised set of equipment (like a 'workshop') for working on information and communication.
- Components of integrated arrangements of devices, tools, services and practices that enable information to be collected, processed, stored and shared with others.
- Components in a comprehensive system of people, information and devices that enables learning, problem solving and higher order collaborative thinking, that is, ICT as key elements underpinning a (sharable) workspace.

2.17. CLIENT/SERVER ARCHITECTURE

Client-Server architecture is an architectural deployment style that describes the separation of functionality into layers with each segment being a tier that can be located on a physically separate computer. They evolved through the component-oriented approach, generally using platform specific methods for communication instead of a message-based approach.

This architecture has different usages with different applications. It can be used in web applications and distributed applications. The strength in particular is when using this architecture over distributed systems. In this course work, I will furthermore invest this through the example of three-tier architecture in web applications.

Structure

Using this architecture the software is divided into 3 different tiers: Presentation tier, Logic tier, and Data tier. Each tier is developed and maintained as an independent tier.

- 1-**Presentation tier**

This is the topmost level of the application. The presentation layer provides the application’s user interface (UI). Typically, this involves the use of Graphical User Interface for smart client interaction, and Web based technologies for browser-based interaction. The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents. It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network.

- **2-Logic tier** (called also business logic, data access tier, or middle tier)

  The logic tier is pulled out from the presentation tier and, as its own layer; it controls an application’s functionality by performing detailed processing. Logic tier is where mission-critical business problems are solved. The components that make up this layer can exist on a server machine, to assist in resource sharing. These components can be used to enforce business rules, such as business algorithms and legal or governmental regulations, and data rules, which are designed to keep the data structures consistent within either specific or multiple databases. Because these middle-tier components are not tied to a specific client, they can be used by all applications and can be moved to different locations, as response time and other rules require.

- **3-Data tier**

  This tier consists of database servers, is the actual DBMS access layer. It can be accessed through the business services layer and on occasion by the user services layer. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance. This layer consists of data access components (rather than raw DBMS connections) to aid in resource sharing and to allow clients to be configured without installing the DBMS libraries and ODBC drivers on each client.

The basic characteristics of client/server architectures are:

- Combination of a client or front-end portion that interacts with the user, and a server or back-end portion that interacts with the shared resource. The client process contains solution-specific logic and provides the interface between the user and the rest of the application system. The server process acts as a software engine that manages shared resources such as databases, printers, modems, or high powered processors.
- The front-end task and back-end task have fundamentally different requirements for computing resources such as processor speeds, memory, disk speeds and capacities, and input/output devices.
- The environment is typically heterogeneous and multivendor. The hardware platform and operating system of client and server are not usually the same.Client and server processes communicate through a well-defined set of standard application program interfaces (API's) and RPC's.

- An important characteristic of client-server systems is scalability. They can be scaled horizontally or vertically. Horizontal scaling means adding or removing client workstations with only a slight performance impact. Vertical scaling means migrating to a larger and faster server machine or multi servers.

2.18. LET US SUM UP

In this unit, you have learnt about the architecture of DBMS, DDL & DML, Database administrator & Database manager, data dictionary and distributed Processing. This knowledge would make you understand the three level architecture of DBMS, various languages used in database development, the players participated in database system management and the concept of ICT placed in database. Thus, the database system architecture unit would have brought you to closer to know the concept of database systems architecture and players of database management.

2.19. UNIT – END QUESTIONS

1. Discuss about the three level architecture of DBMS.
2. Describe about the use of DDL and DML in detail.
3. Write about the role of database administrator & manager in DBMS.

2.20. ANSWER TO CHECK YOUR PROGRESS

1. DBMS architecture helps in design, development, implementation, and maintenance of a database. A database stores critical information for a business. Selecting the correct database architecture helps in quick and secure access to this data. There are mainly three levels of data abstraction: 1. Internal Level: Actual PHYSICAL storage structure and access paths, 2. Conceptual or Logical Level: Structure and constraints for the entire database, 3. External or View level: Describes various user views.

2. Data Definition language (DDL) in DBMS with Examples: Data Definition Language can be defined as a standard for commands through which data structures are defined. It is a computer language that used for creating and modifying the structure of the database objects, such as schemas, tables, views, indexes, etc. Additionally, it assists in storing the metadata

details in the database. Data Manipulation Language (DML) can be defined as a set of syntax elements that are used to manage the data in the database. The commands of DML are not auto-committed and modification made by them is not permanent to the database. It is a computer programming language that is used to perform select, insert, delete and update data in a database. The user requests are assisted by Data Manipulation Language. This language is responsible for all forms of data modification in a database.

3. The data manager is the central software component of the DBMS. It is sometimes referred to as the database control system. One of the functions of the data manager is to convert operations in the user's queries coming directly via the query processor or indirectly via an application program from user's logical view to a physical file system. The data manager is responsible for interfacing with the file system. DBA administers the three levels of database and consultation with the overall user community, sets up the definition of the global view of the various users and applications and is responsible the definition and implementation of the internal level, including the storage structure and access methods to be used for the optimum performance of the DBMS. DBA is responsible for granting permission to the users of the database and stores the profile of each user in the database.

UNIT III – DATABASE MODELS AND IMPLEMENTATION

Structure

UNIT III – DATABASE MODELS AND IMPLEMENTATION

3.1. INTRODUCTION

In this lesson you will be aware with the various data models used to construct the simple database systems. These elements include the various data models like relational data model, Hierarchical model, network data model, object relational model, object oriented model, ER model and associative database model. These elements are used to construct more comprehensive database system. Some of the elements need very detailed information; however, the purpose of this type of data models is to introduce certain.

3.2. OBJECTIVES

After going through this lesson you would be in a positions to

- Various database models used in DBMS.

- ➢ Entity-Relationship Model and its notations.
- ➢ Define relationship and types of relationship.

3.3. DATA MODEL AND TYPES OF DATA

Data modeling is the process of creating a data model for the data to be stored in a Database. This data model is a conceptual representation of

- Data objects
- The associations between different data objects
- The rules.

Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data. Data Models ensure consistency in naming conventions, default values, semantics, and security while ensuring quality of the data.

Data model emphasizes on what data is needed and how it should be organized instead of what operations need to be performed on the data. Data Model is like architect's building plan which helps to build a conceptual model and set the relationship between data items.

3.3.1. Relational Data Model

The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts

1. Attribute: Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
2. Tables – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. Tuple – It is nothing but a single row of a table, which contains a single record.
4. Relation Schema: A relation schema represents the name of the relation with its attributes.
5. Degree: The total number of attributes which in the relation is called the degree of the relation.

6. Cardinality: Total number of rows present in the Table.
7. Column: The column represents the set of values for a specific attribute.
8. Relation instance – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. Relation key - Every row has one, two or multiple attributes, which is called relation key.
10. Attribute domain – Every attribute has some pre-defined value and scope which is known as attribute domain

Operations in Relational Model

Four basic update operations performed on relational database model are,
Insert, update, delete and select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called Relational Integrity Constraints. There are three main integrity constraints,

- Key constraints
- Domain constraints
- Referential integrity constraints

➢ **Key Constraints**

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called key for that relation. If there are more than one such minimal subsets, these are called candidate keys.

➢ **Domain Constraints**

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

➢ **Referential integrity Constraints**

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

3.3.2. Hierarchical Model

A hierarchical model represents the data in a tree-like structure in which there is a single parent for each record. To maintain order there is a sort field which keeps sibling nodes into a

recorded manner. These types of models are designed basically for the early mainframe database management systems, like the Information Management System (IMS) by IBM.

This model structure allows the one-to-one and a one-to-many relationship between two/ various types of data. This structure is very helpful in describing many relationships in the real world; table of contents, any nested and sorted information.

The hierarchical structure is used as the physical order of records in storage. One can access the records by navigating down through the data structure using pointers which are combined with sequential accessing. Therefore, the hierarchical structure is not suitable for certain database operations when a full path is not also included for each record.

Data in this type of database is structured hierarchically and is typically developed as an inverted tree. The "root" in the structure is a single table in the database and other tables act as the branches flowing from the root. The diagram below shows a typical hierarchical database structure.

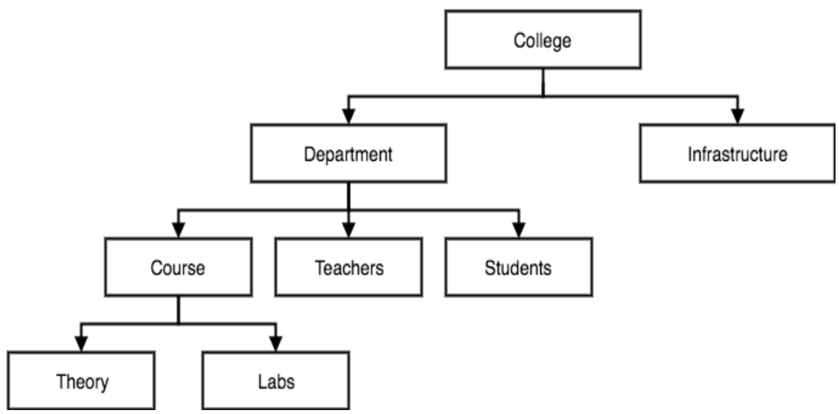


Figure 9: - Hierarchical Data Model

A user can access the data by starting at the root table and working down through the tree to the target data. the user must be familiar with the structure of the database to access the data without any complexity.

Advantages

1. A user can retrieve data very quickly due to the presence of explicit links between the table structures.
2. The referential integrity is built in and automatically enforced due to which a record in a child table must be linked to an existing record in a parent table, along with that if a record deleted in the parent table then that will cause all associated records in the child table to be deleted as well.

Disadvantages

1. When a user needs to store a record in a child table that is currently unrelated to any record in a parent table, it gets difficulty in recording and user must record an additional entry in the parent table.
2. This type of database cannot support complex relationships, and there is also a problem of redundancy, which can result in producing inaccurate information due to the inconsistent recording of data at various sites.

3.3.3. Network Data Model

The Network model replaces the hierarchical tree with a graph thus allowing more general connections among the nodes. The main difference of the network model from the hierarchical model, is its ability to handle many to many (N:N) relations. In other words, it allows a record to have more than one parent. Suppose an employee works for two departments. The strict hierarchical arrangement is not possible here and the tree becomes a more generalized graph - a network. The network model was evolved to specifically handle non-hierarchical relationships. As shown below data can belong to more than one parent. Note that there are lateral connections as well as top-down connections. A network structure thus allows 1:1 (one: one), l: M (one: many), M: M (many: many) relationships among entities.

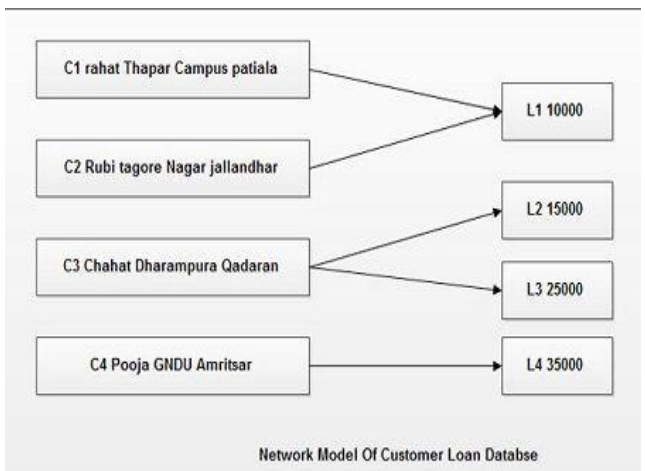


Figure 10: - Example of Network Model

In network database terminology, a relationship is a set. Each set is made up of at least two types of records: an owner record (equivalent to parent in the hierarchical model) and a member record (similar to the child record in the hierarchical model).

The database of Customer-Loan, which we discussed earlier for hierarchical model, is now represented for Network model as shown.

In can easily depict that now the information about the joint loan L1 appears single time, but in case of hierarchical model it appears for two times. Thus, it reduces the redundancy and is better as compared to hierarchical model.

The network model has the following major features:

1. It can represent redundancy in data more efficiently than that in the hierarchical model.
2. There can be more than one path from a previous node to successor node/s.
3. The operations of the network model are maintained by indexing structure of linked list (circular) where a program maintains a current position and navigates from one record to another by following the relationships in which the record participates.
4. Records can also be located by supplying key values.

Advantages

1. Fast data access.
2. It also allows users to create queries that are more complex than those they created using a hierarchical database. So, a variety of queries can be run over this model.

Disadvantages

1. A user must be very familiar with the structure of the database to work through the set structures.
2. Updating inside this database is a tedious task. One cannot change a set structure without affecting the application programs that use this structure to navigate through the data. If you change a set structure, you must also modify all references made from within the application program to that structure.

3.3.4. Object / Relational Model

An Object relational model is a combination of a Object oriented database model and a Relational database model. So, it supports objects, classes, inheritance etc. just like Object Oriented models and has support for data types, tabular structures etc. like Relational data model.

One of the major goals of Object relational data model is to close the gap between relational databases and the object oriented practices frequently used in many programming languages such as C++, C#, Java etc.

History of Object Relational Data Model

Both Relational data models and Object oriented data models are very useful. But it was felt that they both were lacking in some characteristics and so work was started to build a model that was a combination of them both. Hence, Object relational data model was created as a result of research that was carried out in the 1990's.

Advantages of Object Relational model

The advantages of the Object Relational model are:

Inheritance

The Object Relational data model allows its users to inherit objects, tables etc. so that they can extend their functionality. Inherited objects contain new attributes as well as the attributes that were inherited.

Complex Data Types

Complex data types can be formed using existing data types. This is useful in Object relational data model as complex data types allow better manipulation of the data.

Extensibility

The functionality of the system can be extended in Object relational data model. This can be achieved using complex data types as well as advanced concepts of object oriented model such as inheritance.

Disadvantages of Object Relational model

The object relational data model can get quite complicated and difficult to handle at times as it is a combination of the Object oriented data model and Relational data model and utilizes the functionalities of both of them.

3.3.5. Object-Oriented Model

Object oriented data model is based upon real world situations. These situations are represented as objects, with different attributes. These entire objects have multiple relationships between them.

Elements of Object oriented data model

Objects

The real world entities and situations are represented as objects in the Object oriented database model.

Attributes and Method

Every object has certain characteristics. These are represented using Attributes. The behaviour of the objects is represented using Methods.

Class

Similar attributes and methods are grouped together using a class. An object can be called as an instance of the class.

Inheritance

A new class can be derived from the original class. The derived class contains attributes and methods of the original class as well as its own.

The Components of the Object Oriented Data Model

- An object is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model’s entity. More precisely, an object represents only one occurrence of an entity. (The object’s semantic content is defined through several of the items in this list.)
- Attributes describe the properties of an object. For example, a PERSON object includes the attributes Name, Social Security Number, and Date of Birth.

- Objects that share similar characteristics are grouped in classes. A class is a collection of similar objects with shared structure (attributes) and behavior (methods). In a general sense, a class resembles the ER model's entity set. However, a class is different from an entity set in that it contains a set of procedures known as methods. A class's method represents a real-world action such as finding a selected PERSON's name, changing a PERSON's name, or printing a PERSON's address. In other words, methods are the equivalent of procedures in traditional programming languages. In OO terms, methods define an object's behavior.
- Classes are organized in a class hierarchy. The class hierarchy resembles an upside-down tree in which each class has only one parent. For example, the CUSTOMER class and the EMPLOYEE class share a parent PERSON class. (Note the similarity to the hierarchical data model in this respect.)
- Inheritance is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it. For example, two classes, CUSTOMER and EMPLOYEE, can be created as subclasses from the class PERSON. In this case, CUSTOMER and EMPLOYEE will inherit all attributes and methods from PERSON.

Advantages of Object Oriented Data Model

1. Add semantic content
2. Visual presentation includes semantic content
3. Database integrity
4. Both structural and data independence

Disadvantages of Object Oriented Data Model

1. Lack of OODM standards
2. Complex navigational data access
3. Steep learning curve
4. High system overhead slows transactions

3.3.6. Entity-Relationship Model

Entity Relationship Modeling (ER Modeling) is a graphical approach to database design. It uses Entity/Relationship to represent real world objects.

An Entity is a thing or object in real world that is distinguishable from surrounding environment. For example each employee of an organization is a separate entity. Following are some of major characteristics of entities.

- An entity has a set of properties.
- Entity properties can have values.

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- Simple attribute − Simple attributes are atomic values, which cannot be divided further..
- Composite attribute − Composite attributes are made of more than one simple attribute..
- Derived attribute − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database..
- Multi-value attribute − Multi-value attributes may contain more than one values.

These attribute types can come together in a way like −

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

3.3.6.1. Modeling using E-R Diagrams

Entity relationship diagram displays the relationships of entity set stored in a database. In other words, we can say that ER diagrams help you to explain the logical structure of databases. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.

Facts about ER Diagram Model:

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data
- Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities

Why use ER Diagrams?

Here, are prime reasons for using the ER Diagram

- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- ERD is allowed you to communicate with the logical structure of the database to users

Components of the ER Diagram

This model is based on three basic concepts:

- Entities
- Attributes
- Relationships

An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

Relationship is nothing but an association among two or more entities. E.g., Tom works in the Chemistry department.

It is a single-valued property of either an entity-type or a relationship-type. For example, a lecture might have attributes: time, date, duration, place, etc. An attribute is represented by an Ellipse

3.3.6.2. Notation used in E-R Model

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation.

All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of connection. The symbols used for the basic ER constructs are:

- Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.
- Attributes are represented by Ellipses.
- A solid line connecting two entities represents relationships. The name of the relationship is written above the line. Relationship names should be verbs and diamonds sign is used to represent relationship sets.

- Attributes, when included, are listed inside the entity rectangle. Attributes, which are identifiers, are underlined. Attribute names should be singular nouns.
- Multi-valued attributes are represented by double ellipses.
- Directed line is used to indicate one occurrence and undirected line is used to indicate many occurrences in a relation.

The symbols used to design an ER diagram are shown.

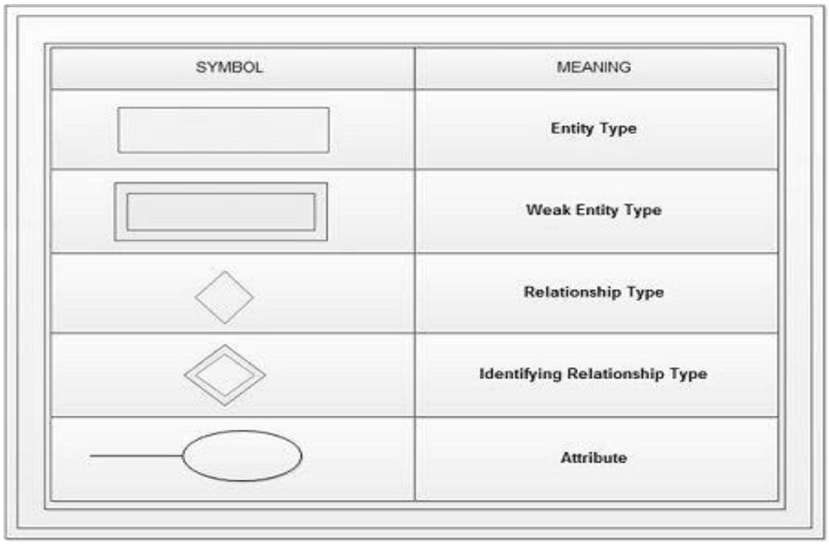


Figure 11: - Symbol used for ER Diagram

3.3.7. Relationships and relationship Types

A relationship, in the context of databases, is a situation that exists between two relational database tables when one table has a foreign key that references the primary key of the other table. Relationships allow relational databases to split and store data in different tables, while linking disparate data items.

There are 3 types of relationships in relational database design. They are:

- One-to-One
- One-to-Many (or Many-to-One)
- Many-to-Many

One-to-One Relationships

A pair of tables bears a one-to-one relationship when a single record in the first table is related to only one record in the second table, and a single record in the second table is related to only one record in the first table.

One-to-Many Relationships

A one-to-many relationship exists between a pair of tables when a single record in the first table can be related to one or more records in the second table, but a single record in the second table can be related to only one record in the first table.

Many-to-Many Relationships

A pair of tables bears a many-to-many relationship when a single record in the first table can be related to one or more records in the second table and a single record in the second table can be related to one or more records in the first table.

3.3.8. Associative database Model

The associative model of data is a data model for database systems. Other data models, such as the relational model and the object data model, are record-based. These models involve encompassing attributes about a thing, such as a car, in a record structure. Such attributes might be registration, colour, make, model, etc. In the associative model, everything which has “discrete independent existence” is modeled as an entity, and relationships between them are modeled as associations.

Associative model has a division property; this divides the real world things about which data is to be recorded in two sorts i.e. between entities and associations. Thus, this model does the division for dividing the real world data to the entities and associations.

The Associative data model is a model for databases unlike any of those we spoke of in prior articles. Unlike the relational model, which is record based and deals with entities and attributes, this model works with entities that have a discreet independent existence, and their relationships are modeled as associations.

The Associative model was bases on a subject-verb-object syntax with bold parallels in sentences built from English and other languages.

The Associative database had two structures, there are a set of items and a set of links that are used to connected them together. With the item structure the entries must contain a unique indication, a type, and a name. Entries in the links structure must also have a unique indicator along with indicators for the related source, subject, object, and verb.

How the Associative Data is Model different?

The Associative model structure is efficient with the storage room fore there is no need to put aside existing space for the data that is not yet available. This differs from the relational model structure. With the relational model the minimum of a single null byte is stored for missing data in any given row. Also some relational databases set aside the maximum room for a specified column in each row.

The Associative database creates storage of custom data for each user, or other needs clear cut and economical when considering maintenance or network resources. When different data needs to be stored the Associative model is able to manage the task more effectively then the relational model.

With the Associative model there are entities and associations. The entity is identified as discrete and has an independent existence, where as the association depends on other things.

The Associative model is designed to store metadata in the same structures where the data itself is stored. This metadata describes the structure of the database and the how different kinds of data can interconnect. Simple data structures need more to transport a database competent of storing the varying of data that a modernized business requires along with the protection and managements that is important for internet implementation.

The Associative model is built from chapters and the user's view the content of the database is controlled by their profile. The profile is a list of chapters. When some links between items in the chapters inside as well as outside of a specific profile exist, those links will not be visible to the user.

3.4. LET US SUM UP

In this unit, you have learnt about the various data model and types of data used in database system development and implementation. This knowledge would make you understand the data models used in database management system such as network model. Hierarchical model, object oriented model, object relational model and relationship of data. Thus, the data models and implementation unit would have brought you to closer to know the concept of data models to represent the data database management system.

3.5. UNIT – END QUESTIONS

1. Identify the various data models used in DBMS.
2. Analyze the use of ER Model in Database design.

3.6. ANSWER TO CHECK YOUR PROGRESS

1. Data modeling is the process of creating a data model for the data to be stored in a Database. The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship. A hierarchical model represents the data in a tree-like structure in which there is a single parent for each record. The Network model replaces the hierarchical tree with a graph thus allowing more general

connections among the nodes. An Object relational model is a combination of a Object oriented database model and a Relational database model. Object oriented data model is based upon real world situations. These situations are represented as objects, with different attributes.

2. Entity Relationship Modeling (ER Modeling) is a graphical approach to database design. It uses Entity/Relationship to represent real world objects. An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. Entities are represented by means of their properties, called attributes. All attributes have values. Entity relationship diagram displays the relationships of entity set stored in a database. All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of connection. A relationship, in the context of databases, is a situation that exists between two relational database tables when one table has a foreign key that references the primary key of the other table.

UNIT IV – FILE ORGANIZATION AND CONVENTIONAL DBMS

Structure

UNIT IV – FILE ORGANIZATION FOR CONVENTIONAL DBMS

4.1. INTRODUCTION

In this lesson you will be aware with the various storage devices and its characteristics used for database storage and file organisation mechanism of DBMS. These device characteristics and file mechanism include the magnetic disk and its performances and various types of file organisation used in various devices.

4.2. OBJECTIVES

After going through this lesson you would be in a positions to

- ➢ Storage devices and its characteristics.
- ➢ Recognize the performances of storage devices.
- ➢ Define files, file organisations and representation.

4.3. STORAGE DEVICES AND CHARACTERISTICS

Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types –

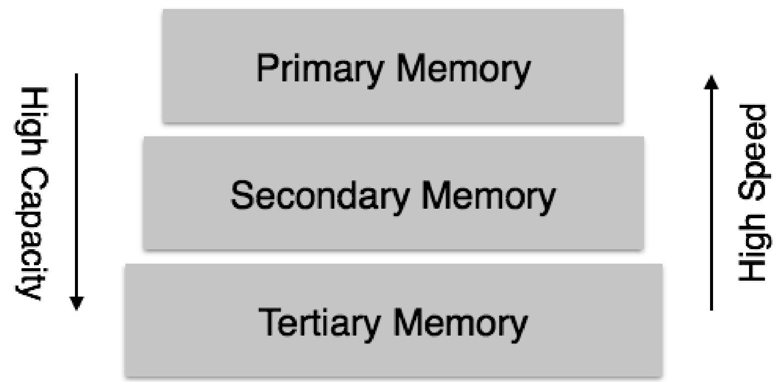


Figure 12: - Storage Devices

- **Primary Storage** − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
- **Secondary Storage** − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.
- **Tertiary Storage** − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

Memory Hierarchy

A computer system has a well-defined hierarchy of memory. A CPU has direct access to it main memory as well as its inbuilt registers. The access time of the main memory is obviously less than the CPU speed. To minimize this speed mismatch, cache memory is introduced. Cache memory provides the fastest access time and it contains data that is most frequently accessed by the CPU.

The memory with the fastest access is the costliest one. Larger storage devices offer slow speed and they are less expensive, however they can store huge volumes of data as compared to CPU registers or cache memory.

4.3.1. Magnetic Disks

A magnetic disk is a storage device that uses a magnetization process to read, write, rewrite and access data. It is covered with a magnetic coating and stores data in the form of tracks, spots and sectors. Hard disks, zip disks and floppy disks are common examples of magnetic disks.

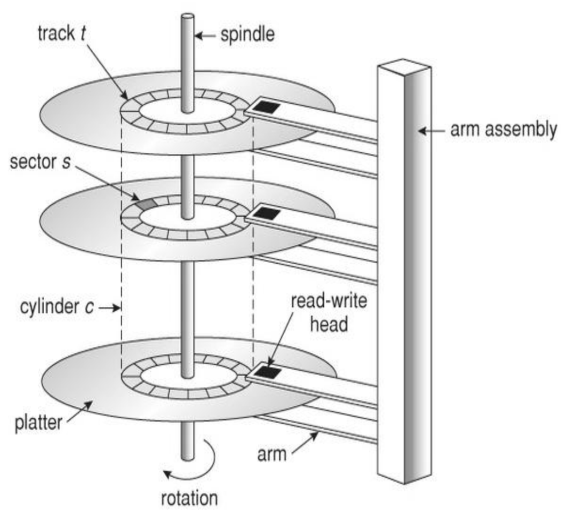


Figure 13: - Magnetic Disk

4.3.2. Physical Characteristics of Disks

A magnetic disk primarily consists of a rotating magnetic surface and a mechanical arm that moves over it. That mechanical arm is used to read from and write data to the disk. The data on a magnetic disk is read and written using a magnetization process. Data is organized on the disk in the form of tracks and sectors, where tracks are the circular divisions of the disk. Tracks are further divided into sectors that contain blocks of data. All read & write operations on the magnetic disk are performed on the sectors.

Magnetic disks have traditionally been used as primary storage in computers. But now with the advent of solid-state drives (SSDs), Flash drives (Pendrive), External hard disks..Magnetic disks are no longer considered the only option, but are still commonly used.

A. Magnetic Read and Write Mechanisms. Write head is has coils of wire around a gapped rectangular doughnut.

Read head uses a partially shielded magneto resistive sensor made of ferromagnetic materials.

B. Data Organization and Formatting.

1. Tracks = concentric rings on the platter, each the width of a head, and separated by the intertrack gap.
2. Sectors = sections of tracks (512 bytes of data, recently 4K) separated by intersector gaps.
3. Tracks on the outside of the platter have a higher angular velocity than tracks closer to the center.
   a. Constant angular velocity layout has the same number of sectors on each track, so outer ones are longer than inner ones. Advantage: easy to find sectors. Disadvantage: Outer sectors have lower density than possible.
   b. Multiple zoned layout has the number of sectors based on the circumference of the tracks. Advantage: makes full use of the density possible, and thus more storage capacity.
4. Formatting creates physical sectors (600 bytes) that also include IDs, CRC, and gaps to help with synching.

C. Physical Characteristics

1. Heads: fixed head (one per track, obsolete), or moveable-head (one per surface) mounted on an arm.
2. Non-removable disk (hard disk) or removable disk (floppy).
3. Single platter, or multiple (2 -10) platters on the same spindle.
4. Single sided, or double sided (usually).
5. Head Mechanisms: 1) contact, e.g. floppy; 2) fixed gap (traditional); and 3) aerodynamic gap (Winchester) relies on foil head riding the air current produced by the spinning platters to keep from touching them.

4.3.3. Performance Measures of Disks

1. Seek time = average time it takes to position a head over the correct track. 4-13ms. Zero on fixed head drives.
2. Rotational delay = average time it takes a sector to reach the head. 7200 RPM = 4ms, 10000 RPM = 3 ms
3. Access time = seek time + rotational delay.

4. Transfer time = time to read or write one sector once it has reached the head, 100 - 300MB/s

4.3.4. Optimization of Disk-Block Access

1. Data is transferred between disk and main memory in units called blocks.
2. A block is a contiguous sequence of bytes from a single track of one platter.
3. Block sizes range from 512 bytes to several thousand.
4. The lower levels of file system manager covert block addresses into the hardware-level cylinder, surface, and sector number.
5. Access to data on disk is several orders of magnitude slower than is access to data in main memory. Optimization techniques besides buffering of blocks in main memory.
    - Scheduling: If several blocks from a cylinder need to be transferred, we may save time by requesting them in the order in which they pass under the heads. A commonly used disk-arm scheduling algorithm is the elevator algorithm.
    - File organization. Organize blocks on disk in a way that corresponds closely to the manner that we expect data to be accessed. For example, store related information on the same track, or physically close tracks, or adjacent cylinders in order to minimize seek time. IBM mainframe OS's provide programmers fine control on placement of files but increase programmer's burden.

      UNIX or PC OSs hide disk organizations from users. Over time, a sequential file may become fragmented. To reduce fragmentation, the system can make a back-up copy of the data on disk and restore the entire disk. The restore operation writes back the blocks of each file continuously (or nearly so). Some systems, such as MS-DOS, have utilities that scan the disk and then move blocks to decrease the fragmentation.
    - Nonvolatile writes buffers. Use nonvolatile RAM (such as battery-back-up RAM) to speed up disk writes drastically (first write to nonvolatile RAM buffer and inform OS that writes completed).
    - Log disk. Another approach to reducing write latency is to use a log disk, a disk devoted to writing a sequential log. All access to the log disk is sequential, essentially eliminating seek time, and several consecutive blocks can be written at once, making writes to log disk several times faster than random writes.

4.4. FILE ORGANIZATION

A file is a collection of or log of records. Having stored the records in a file it is necessary to access these records using either a primary or secondary key. The type and

frequency of access required determines the type of file organization to be used for a given set of records. A File is organized logically as a sequence of records. These records are mapped onto disk blocks. Files are provided as a basic construct in operating systems, representing logical data models in terms of files. Although blocks are of a fixed size determined by the physical properties of the disk and by the operating system, record sizes vary. In a relational database, tuples of distinct relations are generally of different sizes.

One approach to mapping the database to files is to use several files and to store records of only one fixed length in any given file. An alternative is to structure our files so that we can accommodate multiple lengths for records; however, files of fixed length records are easier to implement than are files of variable length records. Many of the techniques used for the former can be applied to the variable length case.

Storing the files in certain order is called file organization. The main objective of file organization is

- Optimal selection of records i.e.; records should be accessed as fast as possible.
- Any insert, update or delete transaction on records should be easy, quick and should not harm other records.
- No duplicate records should be induced as a result of insert, update or delete
- Records should be stored efficiently so that cost of storage is minimal.

There are various methods of file organizations. These methods may be efficient for certain types of access/selection meanwhile it will turn inefficient for other selections.

Various methods have been introduced to Organize files. These particular methods have advantages and disadvantages on the basis of access or selection. Some types of File Organizations are:

- Sequential File Organization.
- Heap File Organization.
- Hash File Organization.
- B+ Tree File Organization.
- Clustered File Organization.

4.4.1. Fixed-Length Records

As an example, let us consider a file of account records for our bank database .Each record of this file is defined (in pseudo code) as:

```
type deposit = record
        Account _number char (10);
        Branch_name char (22);
        Balance numeric (12, 2);
end
```

If we assume that each character occupies 1 byte and that numeric (12, 2) occupies 8 bytes, our account record is 40 bytes long. A simple approach is to use the first 40 bytes for the first record, the next 40 bytes for the second record and so on.
However there are two main problems with this simple approach.

1. It is difficult to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of the file, or we must have a way of marking deleted records so that they can be ignored.
2. Unless the block size happens to be a multiple of 40(which is unlikely), some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two block accesses to read or write such a record.

When a record is deleted, we could move the record that came after it into the space formerly occupied by the deleted record, and so on, until every record following the deleted record has been moved ahead. Such an approach requires moving a large number of records. It might be easier simply to move the final record of the file into the space occupied by the deleted record.

It is undesirable to move records to occupy the space freed by the deleted record, since doing so requires additional block accesses. Since insertions tend to be more frequent than deletions, it is acceptable to leave open the space occupied by the deleted record, and to wait for a subsequent insertion before reusing the space. A simple marker on the deleted record is not sufficient, since it is hard to find this available space when an insertion is being done. Thus we need to introduce an additional structure.

At the beginning of the file, we allocate a certain number of bytes as a file header. The header will contain a variety of information about the file.

For now, all we need to store there is the address of the first record whose contents are deleted. We use this first record to store the address of the second available record, and so on. Intuitively we can think of these stored addresses as pointers, since they point to the location of a record. The deleted records thus form a linked list, which is often referred to as a free list.

On insertion of a new record, we use the record pointed to by the header. We change the header pointer to point to the next available record. If no space is available, we add the new record to the end of the file.

Insertion and deletion for files of fixed length records are simple to implement, because the space made available by a deleted record is exactly the space needed to insert a record. If we allow records of variable length in a file, this match no longer holds. An inserted record may not fit in the space left free by a deleted record, or it may fill only part of that space.

4.4.2. Variable-Length Records

Variable length records arise in the database systems in several ways.

- Storage of multiple record types in a file
- Record types that allow variable lengths for one or more fields.
- Record types that allow repeating fields, such as arrays or multisets.

Different techniques for implementing variable length records exist.

The slotted page structure is commonly used for organizing records within a block. There is a header at the beginning of each block, containing the following information.

1. The number of record entries in the header.
2. The end of free space in the block.
3. An array whose entries contain the location and size of each record.

The actual records are allocated contiguously in the block, starting from the end of the block. The free space in the block is contiguous, between the final entry in the header array, and the first record. If a record is inserted, space is allocated for it at the end of free space, and an entry containing its size and location is added to the header.

If a record is deleted, the space that it occupies is freed, and its entry is set to deleted (its size is set to -1, for example). Further the records in the block before the deleted records are moved, so that the free space created by the deletion gets occupied, and all free space is again between the final entry in the header array and the first record. The end of free space pointer in the header is appropriately updated as well. Records can be grown or shrunk by similar techniques, as long as there is space in the block. The cost of moving the records is not too high, since the size of a block is limited: a typical value is 4 kilobytes.

The slotted page structure requires that there be no pointers that point directly to records. Instead, pointers must point to the entry in the header that contains the actual location of the record. This level of indirection allows records to be moved to prevent fragmentation of space inside a block, while supporting indirect pointers to the record.

Databases often store data that can be much larger than a disk block .For instance, an image or an audio recording may be multiple megabytes in size, while a video object may be multiple gigabytes in size. Recall that SQL supports the types blob and clob, which store binary and character large objects.

Most relational databases restrict the size of a record to be no larger than the size of a block, to simplify buffer management and free space management. Large objects are often stored in a special file (or collection of files) instead of being stored with the other (short) attributes of records in which they occur. Large objects are often represented using B+ - tree file organizations.

4.4.3. Organization of records in files

So far, we have studied how records are represented in a file structure. A relation is a set of records. Given a set of records; the next question is how to organize them in a file. Several of the possible ways of organizing records in files are:

- **Heap File Organization: -** Any record can be placed anywhere in the file where there is space for a record. There is no ordering of records. Typically, there is a single file for each relation.

- **Sequential File: -** Organization Records are stored in sequential order, according to the value of a "search key" of each record.
- **Hashing File Organization: -** A hash function is computed on some other attribute of each record. The result of the hash function specifies in which block of the file the record should be placed.

4.4.4. Sequential file Organization

In a sequential file, records are maintained in the logical sequence of their primary key values. The processing of a sequential file is conceptually simple but inefficient for random access. A sequential file could be stored on a sequential storage device such as a magnetic tape.

Search for a given record in a sequential file requires, an average access to half the records in the file. A binary or logarithmic search technique may also be used to search for a record.

Updating usually requires the creation of a new file. To maintain file sequence records are copied to the point where amendment is required. The changes are then made and copied to the new file. Following this, the remaining records in the original file are copied to the new file, thus creating an automatic back-up copy.

The basic advantage offered by a sequential file is the ease of access to the next record, the simplicity of the organization and the absence of auxiliary data structures. To reduce the cost per update, all updates are batched, sorted in order of sequential file and the file is updated in single pass, such a file containing updates to be made to sequential file is sometimes referred to as 'Transaction File'

Thus a sequential file is designed for efficient processing of records in sorted order based on some search key. A search key is any attribute or set of attributes; it need not be the primary key, or even a super key. To permit fast retrieval of records in search key order, we chain together records by pointers. The pointer in each record points to the next record in the search order. Furthermore, to minimize the number of block accesses in sequential file processing, we store records physically in search key order, or as close to search key order as possible.

The sequential file organization allows records to be read in sorted order, that can be useful for display purposes as well as for certain query processing algorithms.

It is difficult, however to maintain physical sequential order as records are inserted and deleted, since it is costly to move many records as a result of a single insertion or deletion. We can manage deletion by using pointer chains, as we saw previously. For insertion, we apply the following rules.

1. Locate the record in the file that comes before the record to be inserted in search key order.
2. If there is a free record (that is, space left after a deletion) within the same block as this record, insert the new record there. Otherwise, insert the new record in an overflow

block. In either case, adjust the pointers so as to chain together the records in search key order.

4.4.5. Indexed Sequential Access Method (ISAM)

The retrieval of a record from a sequential file, on average, requires access to half the records in the file, making such enquiries not only inefficient but very time consuming for large file. To improve the query response time of a sequential file, a type of indexing technique can be added.

An index is a set of pairs. Indexing associates a set of objects to a set of orderable quantities, which are usually smaller in number or their properties, provide a mechanism for faster search. The purpose of indexing is to expedite the search process.

'Indexes created from a sequential (or sorted) set of primary keys are referred to as index sequential'. Although the indices and the data blocks are held together physically, we distinguish between them logically. We shall use the term index file to describe the indexes and data file to refer to the data records. The index is usually small enough to be read into the processor memory.

A sequential (or sorted on primary keys) file that is indexed is called an index sequential file. The index provides for random access to records, while the sequential nature of the file provides easy access to the subsequent records as well as sequential processing. An additional feature of this file system is the overflow area. This feature provides additional space for record addition without necessitating the creation of a new file.

In index sequential organization, it is the usual practice to have a hierarchy of indexes with the lowest level index pointing to the records while the higher level ones point to the index below them.

Updates to an index sequential file may entail modifications to the index in addition to the file The index file can be simplified or its storage requirements reduced if only the address part of the pair is held in the index. This however necessitates holding the address of every possible key in the key range, including addresses of records not in file. The addresses of nonexistent records can be set to an impossibly high or low value to indicate their absence from the file. If the number of such missing records in the range of stored key values is small, the saving obtained by not storing the key is considerable.

4.4.6. Virtual Storage Access Method (VSAM)

Virtual Storage Access Method (VSAM) is high performance access method and data set organization, which organizes and maintains data via a catalog structure. It utilizes virtual storage concept and can protect datasets at various levels by giving passwords. VSAM can be used in COBOL programs like physical sequential files. VSAM are the logical datasets for storing records. Files can be read sequentially and randomly in VSAM. It is an improved way of

storing data which overcomes some of the limitations of conventional file systems like Sequential Files.

Characteristics of VSAM

Following are the characteristics of VSAM −

- VSAM protects data against unauthorized access by using passwords.
- VSAM provides fast access to data sets.
- VSAM has options for optimizing performance.
- VSAM allows data set sharing in both batch and online environment.
- VSAM are more structured and organized in storing data.
- Free space is reused automatically in VSAM files.

Limitations of VSAM

The only limitation of VSAM is that it cannot be stored on TAPE volume. It is always stored on DASD space. It requires a number of cylinders to store the data which is not cost-effective.

VSAM consists of following components −

- VSAM Cluster.
- Control Area.
- Control Interval.

VSAM Cluster

VSAM are the logical datasets for storing records and are known as clusters. A cluster is an association of the index, sequence set and data portions of the dataset. The space occupied by a VSAM cluster is divided in contiguous areas called Control Intervals. We will discuss about control intervals later in this module.

There are two main components in a VSAM cluster −

- Index Component contains the index part. Index records are present in Index component. Using index component VSAM is able to retrieve records from the data component.
- Data Component contains the data part. Actual data records are present in Data component.

Control Interval

Control Intervals (CI) in VSAM are equivalent to blocks for non-VSAM data sets. In non-VSAM methods, the unit of data is defined by the block. VSAM works with logical data area which is known as Control Intervals.

Control Intervals are the smallest unit of transfer between a disk and the operating system. Whenever a record is retrieved directly from the storage, the entire CI containing the record is read into VSAM Input-Output buffer. The desired record is then transferred to work area from VSAM buffer.

Control Interval consists of −

- Logical Records.
- Control information fields.

- Free Space.

When a VSAM dataset is loaded, control intervals are created. The default Control Interval size is 4K bytes and it can extend up to 32K bytes.

Control Area

A Control Area (CA) is formed by putting together two or more Control Intervals. A VSAM dataset is composed of one or more Control Areas. The size of VSAM is always a multiple of its Control Area. VSAM files are extended in units of Control Areas.

4.5. LET US SUM UP

In this unit, you have learnt about the storage devices and various file organisation mechanisms implemented in database management system. This knowledge would make you understand the magnetic disk usage, characteristics and performance of device and various file organisations like sequential file, hash file, heap file and organisation and clustered file also. Thus, the file organisations for conventional DBMS unit would have brought you to closer to know the concept of file organisation to represent the data in the database management system.

4.6. UNIT – END QUESTIONS

1. Examine the use of magnetic disk and its characteristics and performance.
2. Write about the various file organisation mechanisms in detail.

4.7. ANSWER TO CHECK YOUR PROGRESS

1. Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types – Primary Storage, Secondary Storage and Tertiary Storage. A magnetic disk is a storage device that uses a magnetization process to read, write, rewrite and access data. It is covered with a magnetic coating and stores data in the form of tracks, spots and sectors. Hard disks, zip disks and floppy disks are common examples of magnetic disks. A magnetic disk primarily consists of a rotating magnetic surface and a mechanical arm that moves over it. That mechanical arm is used to read from and write data to the disk. The data on a magnetic disk is read and written using a magnetization process. Data is organized on the disk in the form of tracks and sectors, where tracks are the circular divisions of the disk.

Tracks are further divided into sectors that contain blocks of data. All read & write operations on the magnetic disk are performed on the sectors.

2. A file is a collection of or log of records. Having stored the records in a file it is necessary to access these records using either a primary or secondary key. The type and frequency of access required determines the type of file organization to be used for a given set of records. A File is organized logically as a sequence of records. These records are mapped onto disk blocks. Files are provided as a basic construct in operating systems, representing logical data models in terms of files. Although blocks are of a fixed size determined by the physical properties of the disk and by the operating system, record sizes vary. In a relational database, tuples of distinct relations are generally of different sizes. Various methods have been introduced to Organize files. These particular methods have advantages and disadvantages on the basis of access or selection. Some types of File Organizations are: Sequential File Organization, Heap File Organization, Hash File Organization, B+ Tree File Organization and Clustered File Organization.

UNIT V – RELATIONAL DATABASE MANAGEMENT SYSTEM

Structure

UNIT V – RDBMS

5.1. INTRODUCTION

As an end in itself, understanding the relational database management system concepts and terms is important to enable you learn more about database and relational database after the course is over. Without understanding these concepts and terms, you will have difficulty discussing relational DBMS ideas with others, and will have difficulty in reading the technical literature. Since computer science is rapidly evolving new database concepts and since database issues are important in many areas of computer science the ability to learn more quickly is important to maintaining your technical edge.

5.2. OBJECTIVES

After going through this lesson you would be able to

- Define relational model and relational DBMS.
- Explain query optimization and system catalog.

5.3. AN INFORMAL LOOK AT THE RELATIONAL MODEL

The relational model frees the user from the details of storage structures and access methods. It is also conceptually simple and more importantly based on sound theoretical principles that provide formal tools to tackle problems arising in database design and maintenance.

In this model, the relation is the only construct required to represent the associations among the attributes of entity as well as relationships among different entities. One of the main reasons for introducing this model was to increase the productivity of the application programmer by eliminating the need to change application program when a change is made to the database. Users need not know exact physical structures to use the database and are protected from any changes made to these structures. They are however still required to know how the data has been positioned into the various relations.

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represent records and columns represent the attributes.

Tuple − A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema − A relation schema describes the relation name (table name), attributes, and their names.

Relation key − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain − Every attribute has some pre-defined value scope, known as attribute domain.

5.3.1. Relational Database Management System

RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

A relational database refers to a database that stores data in a structured format, using rows and columns. This makes it easy to locate and access specific values within the database. It is "relational" because the values within each table are related to each other. Tables may also be related to other tables. The relational structure makes it possible to run queries across multiple tables at once.

While a relational database describes the type of database an RDMBS manages, the RDBMS refers to the database program itself. It is the software that executes queries on the data, including adding, updating, and searching for values. An RDBMS may also provide a visual representation of the data. For example, it may display data in tables like a spreadsheet, allowing you to view and even edit individual values in the table. Some RDMBS programs allow you to create forms that can streamline entering, editing, and deleting data.

Most well known DBMS applications fall into the RDBMS category. Examples include Oracle Database, MySQL, Microsoft SQL Server, and IBM DB2. Some of these programs support non-relational databases, but they are primarily used for relational database management.

What is a table?

The data in an RDBMS is stored in database objects which are called as tables. This table is basically a collection of related data entries and it consists of numerous columns and rows.

What is a field?

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table.

What is a column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

What is a NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

5.3.2. RDBMS Properties

An RDBMS is easily accessible. You execute commands in them Structured Query Language (SQL) to manipulate data. SQL is the international Standards Organization (ISO) standard language for interacting with a RDBMS.

An RDBMS provides full data independence. The organization of the data is independent of the applications that use it. You do not need to specify the access routes to tables or know how data is physically arranged in a database.

A relational database is a collection of individual, named objects. The basic unit of data storage in a relational database is called a table. A table consists of rows and columns used to store values. For access purpose, the order of rows and columns is insignificant. You can control the access order as required.

When querying the database, you use conditional operations such as joins and restrictions. A join combines data from separate database rows. A restriction limits the specific rows returned by a query.

An RDBMS enables data sharing between users. At the same time, you can ensure consistency of data across multiple tables by using integrity constraints. An RDBMS uses various types of data integrity constraints. These types include entity, column, referential and user- defined constraints. The constraint, entity, ensures uniqueness of rows, and the constraint column ensures consistency of the type of data within a column. The other type, referential, ensures validity of foreign keys, and user-defined constraints are used to enforce specific business rules.

An RDBMS minimizes the redundancy of data. This means that similar data is not repeated in multiple tables.

5.3.3. The Entity-Relationship Model

ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.

It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.

In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

ER models are readily translated to relations. ER models, also called an ER schema, are represented by ER diagrams.

ER modelling is based on two concepts:

- Entities, defined as tables that hold specific information (data)
- Relationships, defined as the associations or interactions between entities

An entity is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be

- An object with physical existence (e.g., a lecturer, a student, a car)
- An object with conceptual existence (e.g., a course, a job, a position)

Entities can be classified based on their strength. An entity is considered weak if its tables are existence dependent.

- That is, it cannot exist without a relationship with another entity
- Its primary key is derived from the primary key of the parent entity

An entity is considered strong if it can exist apart from all of its related entities.

- Kernels are strong entities.

- A table without a foreign key or a table that contains a foreign key that can contain nulls is a strong entity

Another term to know is entity type which defines a collection of similar entities.

An entity set is a collection of entities of an entity type at a particular point of time. In an entity relationship diagram (ERD), an entity type is represented by a name in a box.

5.4. OVERVIEW OF RELATIONAL QUERY OPTIMIZATION

A query can be processed based on indexes and joins, The query optimizer uses these two techniques to determine which process or expression to consider for evaluating the query.

Cost Based Optimization

This is based on the cost of the query. The query can use different paths based on indexes, constraints, sorting methods etc. This method mainly uses the statistics like record size, number of records, number of records per block, number of blocks, table size, whether whole table fits in a block, organization of tables, uniqueness of column values, size of columns etc.

Heuristic Optimization

This method is also known as rule based optimization. This is based on the equivalence rule on relational expressions; hence the number of combination of queries get reduces here. Hence the cost of the query too reduces.

This method creates relational tree for the given query based on the equivalence rules. These equivalence rules by providing an alternative way of writing and evaluating the query, gives the better path to evaluate the query. This rule need not be true in all cases. It needs to be examined after applying those rules. The most important set of rules followed in this method is listed below:

- Perform all the selection operation as early as possible in the query. This should be first and foremost set of actions on the tables in the query. By performing the selection operation, we can reduce the number of records involved in the query, rather than using the whole tables throughout the query.
- Perform all the projection as early as possible in the query. This is similar to selection but will reduce the number of columns in the query.
- Next step is to perform most restrictive joins and selection operations. When we say most restrictive joins and selection means, select those set of tables and views which will result in comparatively less number of records. Any query will have better performance when tables with few records are joined. Hence throughout heuristic method of optimization, the rules are formed to get less number of records at each stage, so that query performance is better. So is the case here too.

- Sometimes we can combine above heuristic steps with cost based optimization technique to get better results.

5.4.1. System Catalog in a Relational DBMS

. The system catalogue is a collection of tables and views that contain important information about a database. It is the place where a relational database management system stores schema metadata, such as information about tables and columns, and internal bookkeeping information. A system catalogue is available for each database.

Information in the system catalogue defines the structure of the database. For example, the DDL (data dictionary language) for all tables in the database is stored in the system catalogue.

Most system catalogues are copied from the template database during database creation, and are thereafter database-specific. A few catalogues are physically shared across all databases in an installation; these are marked in the descriptions of the individual catalogues.

The system catalogue for a database is actually part of the database. Within the database are objects, such as tables, indexes, and views. The system catalogue is basically a group of objects that contain information that defines other objects in the database, the structure of the database itself, and various other significant information.

The system catalogue may be divided into logical groups of objects to provide tables that are accessible by not only the database administrator, but by any other database user as well. A user typically queries the system catalogue to acquire information on the user's own objects and privileges, whereas the DBA needs to be able to inquire about any structure or event within the database. In some implementations, there are system catalogue objects that are accessible only to the database administrator.

5.4.2. Information stored in the System Catalog

In database management systems, a file defines the basic organisation of a database. A data dictionary contains a list of all the files in the database, the number of records in each file, and the names and types of each field. Most database management systems keep the data dictionary hidden from users to prevent them from accidentally destroying its contents.

The information stored in a catalogue of an RDBMS includes:

- The relation names,
- Attribute names,
- Attribute domains (data types),
- Descriptions of constraints (primary keys, secondary keys, foreign keys, NULL/ NOT NULL, and other types of constraints),
- Views, and
- Storage structures and indexes (index name, attributes on which index is defined, type of index etc).

Security and authorization information is also kept in the catalogue, which describes:

- Authorized user names and passwords,
- Each user's privilege to access specific database relations and views,
- The creator and owner of each relation. The privileges are granted using GRANT command.

The system catalogue can also be used to store some statistical and descriptive information about relations. Some such information can be:

- Number of tuples in each relation,
- The different attribute values,
- Storage and access methods used in relation.

All such information finds its use in query processing. In relational DBMSs the catalogue is stored as relations. The DBMS software is used for querying, updating, and maintaining the catalogue. This allows DBMS routines (as well as users) to access. The information stored in the catalogue can be accessed by the DBMS routines as well as users upon authorization with the help of the query language such as SQL.

5.4.3. How Catalogs are Stored

Hence, every database stores every information about its objects. These informations can be it structure, definition, purpose, storage, number of columns and records, dependencies, access rights, owner etc. Basically these are the useful information about the data in the database. They are also called as metadata. These metadata are also stored as rows and columns of a table. Collection of these metadata is stored in the data dictionary or system catalog. Usually, system catalogs are accessed by the DBMS to perform various transactions and data dictionary has the user accessible views that are accessed by the developers/ designers/ users.

It is a database about the database objects. It can exist in the same database or it can be completely a separate database. If it is a separate database, then there should be a process to update the table in system catalog as when object of the database changes. It should be in sync with database objects. This helps the developer/ user to get the quick information about the database objects.

System catalog also contains various tables and views. Since it contains the information about the database objects, users are not allowed to modify the details in it. It is automatically inserted/ updated/ deleted by the RDBMS. User has no access rights on them. But users/developer will need to see the details in them to understand the objects. Sometimes developer wants to know the exact column name and the table in which the records that he is fetching exist. In these cases, he has to fetch in the data dictionary or system catalog. For this purpose views on tables in the data dictionary are created and given SELECT rights to the users. Data dictionary tables are created in the SYS schema and do not have any access privileges for the user. Only the RDBMS access it. The views on these tables are created in SYSTEM schema and users are given only SELCT rights. They cannot modify these views.

5.5. LET US SUM UP

In this unit, you have learnt about the basic concepts of relational database and its properties, and query optimization and system catalogs are discussed. This knowledge would make you understand the difference between DBMS and RDBMS, elaborate the properties of RDBMS, relational query optimization and system catalog strategies are identified. Thus, the RDBMS unit would have brought you to closer to know the concept of relational database.

5.6. UNIT – END QUESTIONS

1. Describe about the concept of relational DBMS and its properties.
2. Discuss about the relational query optimization and system catalog in detail.

5.7. ANSWER TO CHECK YOUR PROGRESS

1. Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency. RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

   A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd. A relational database refers to a database that stores data in a structured format, using rows and columns. This makes it easy to locate and access specific values within the database. It is "relational" because the values within each table are related to each other. Tables may also be related to other tables. The relational structure makes it possible to run queries across multiple tables at once.

2. A query can be processed based on indexes and joins, The query optimizer uses these two techniques to determine which process or expression to consider for evaluating the query. The query can use different paths based on indexes, constraints, sorting methods etc. This method mainly uses the statistics like record size, number of records, number of records per

block, number of blocks, table size, whether whole table fits in a block, organization of tables, uniqueness of column values, size of columns etc.

The system catalogue is a collection of tables and views that contain important information about a database. It is the place where a relational database management system stores schema metadata, such as information about tables and columns, and internal bookkeeping information. A system catalogue is available for each database.

UNIT VI – SQL - I

Structure

UNIT VI – SQL - I

6.1. INTRODUCTION

As you know by now, relational database is an new era for managing the data in effective way. The queries are playing a vital role in database system and database management. In this lesson, you have learnt about the categories of SQL commands, functions and various facilities are identified. An SQL is a query language used for managing the relational data.

6.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain categories of SQL Commands.
- Define functions and updating the database.
- Define data definition facilities.

6.3. CATEGORIES OF SQL COMMANDS

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an ANSI (American National Standards Institute) standard language, but there are many different versions of the SQL language.

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task. There are various components included in this process. These components are –

- Query Dispatcher.
- Optimization Engines.
- Classic Query Engine.
- SQL Query Engine, etc.

A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

SQL commands are instructions, coded into SQL statements, which are used to communicate with the database to perform specific tasks, work, functions and queries with data.

SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users. SQL commands are grouped into four major categories depending on their functionality:

- Data Definition Language (DDL) - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.
- Data Manipulation Language (DML) - These SQL commands are used for storing, retrieving, modifying, and deleting data. These Data Manipulation Language commands are: SELECT, INSERT, UPDATE, and DELETE.

- Transaction Control Language (TCL) - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.
- Data Control Language (DCL) - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

6.3.1. Data Definition

A data definition language (DDL) is a computer language used to create and modify the structure of database objects in a database. These database objects include views, schemas, tables, indexes, etc.

This term is also known as data description language in some contexts, as it describes the fields and records in a database table.

The present database industry incorporates DDL into any formal language describing data. However, it is considered to be a subset of SQL (Structured Query Language). SQL often uses imperative verbs with normal English such as sentences to implement database modifications. Hence, DDL does not show up as a different language in an SQL database, but does define changes in the database schema.

Commonly used DDL in SQL querying are:

- **CREATE:** This command builds a new table and has a predefined syntax. The CREATE statement syntax is CREATE TABLE [table name] ([column definitions]) [table parameters]. CREATE TABLE Employee (Employee Id INTEGER PRIMARY KEY, First name CHAR (50) NULL, Last name CHAR (75) NOT NULL).

  Syntax: -

```
CREATE TABLE <table_name>
(   column_name1 datatype,
    column_name2 datatype,
    .
    .
    .
    column_name_n datatype
);
```

  For example,

```
CREATE TABLE employee
(
    empid INT,
    ename CHAR,
    age INT,
    city CHAR(25),
```

phone_no VARCHAR(20)

);

- **ALTER:** An alter command modifies an existing database table. This command can add up additional column, drop existing columns and even change the data type of columns involved in a database table. An alter command syntax is ALTER object type object name parameters. ALTER TABLE Employee ADD DOB Date.

  Syntax: -

  ALTER TABLE <table_name>
  ADD <column_name datatype>; **OR**
  ALTER TABLE <table_name>
  CHANGE <old_column_name> <new_column_name>; **OR**
  ALTER TABLE <table_name>
  DROP COLUMN <column_name>;

  For Example,

  ALTER TABLE employee
  ADD (address varchar2(50)); **OR**
  ALTER TABLE employee
  CHANGE (phone_no) (contact_no); **OR**
  ALTER TABLE employee
  DROP COLUMN age;

- **DROP:** A drop command deletes a table, index or view. Drop statement syntax is DROP object type object name. DROP TABLE Employee.

  Syntax: -

  DROP TABLE <table_name>; **OR**
  DROP DATABASE <database_name>;

  For Example,

  DROP TABLE employee; **OR**
  DROP DATABASE employees;

6.3.2. Data Manipulation Statements

Data Manipulation Language commands to exercise data operations in the database. Data operations can be populating the database tables with the application or business data, modifying the data and removing the data from the database, whenever required. Besides the data operations, there are set of commands which are used to control these operations. These commands are grouped as Transaction Control Language.

There are three types of DML statements involved in a logical SQL transaction namely, Insert, Update, Delete and Merge.

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a transaction.

The statements you use to add, change, or delete data are called data manipulation statements, which are a subset of the data manipulation language (DML) statements part of ANSI SQL. The main DML statements are:

- INSERT statement   Adds new rows to a table or view.
- UPDATE statement   The UPDATE statement changes rows in a set of tables or views.
- DELETE statement   The DELETE statement removes rows from a set of tables or views.

INSERT statement

The INSERT command is used to store data in tables. The INSERT command is often used in higher-level programming languages

There are two different forms of the INSERT command. The first form is used if a new row will have a value inserted into each column of the row. The second form of the INSERT command is used to insert rows where some of the column data is unknown or defaulted from business logic. This form of the INSERT command requires that you specify column names for which data are being stored.

Syntax: -

```
INSERT INTO table VALUES (column1 value, column2 value, ...);
```

UPDATE statement

The UPDATE command modifies the data stored in a column.It can update single or multiple rows at a time depending on the result set filtered by conditions specified in WHERE clause. Note that updating columns is different from altering columns. Earlier in this chapter, you studied the ALTER command. The ALTER command changes the table structure, but leaves the table data unaffected. The UPDATE command changes data in the table, not the table structure.

Syntax: -

```
UPDATE table SET column = value [, column = value ...][WHERE condition]
```

DELETE statement

The DELETE command is one of the simplest of the SQL statements. It removes one or more rows from a table. Multiple table delete operations are not allowed in SQL.The syntax of the DELETE command is as below.

```
DELETE FROM table_name [WHERE condition];
```

6.3.3. SELECT

SQL is a comprehensive database language. SQL, pronounced Sequel or simply S-Q-L, is a computer programming language used for querying relational databases following a nonprocedural approach. When you extract information from a database using SQL, this is termed querying the database.

A relational database is implemented through the use of a Relational Database Management System (RDBMS). An RDBMS performs all the basic functions of the DBMS software mentioned above along with a multitude of other functions that make the relational model easier to understand and to implement. RDBMS users manipulate data through the use of a special data manipulation language. Database structures are defined through the use of a data definition language. The commands that system users execute in order to store and retrieve data can be entered at a terminal with an RDBMS interface by typing the commands, or entered through use of some type of graphical interface. The DBMS then processes the commands.

Capabilities of the SELECT Statement

Data retrieval from data base is done through appropriate and efficient use of SQL. Three concepts from relational theory encompass the capability of the SELECT statement: projection, selection, and joining.

- Projection: A project operation selects only certain columns (fields) from a table. The result table has a subset of the available columns and can include anything from a single column to all available columns.
- Selection: A select operation selects a subset of rows (records) in a table (relation) that satisfy a selection condition. The ability to select rows from out of complete result set is called Selection. It involves conditional filtering and data staging. The subset can range from no rows, if none of the rows satisfy the selection condition, to all rows in a table.
- Joining: A join operation combines data from two or more tables based on one or more common column values. A join operation enables an information system user to process the relationships that exist between tables. The join operation is very powerful because it allows system users to investigate relationships among data elements that might not be anticipated at the time that a database is designed.

6.3.3.1. The Basic Form

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax: -

The basic syntax of the SELECT statement is as follows −

SELECT column1, column2, … columnN FROM table_name;

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

SELECT * FROM table_name;

The SQL WHERE clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it

returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc.,

6.3.3.2. Sub Queries

The most commonly used SQL command is SELECT statement. SQL SELECT statement is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. To create a simple SQL SELECT Statement, you must specify the column(s) name and the table name. The whole query is called SQL SELECT Statement.

Syntax: -

```
SELECT * | {[DISTINCT] column_list | expression [alias],...}
FROM table-name
[WHERE condition]
[GROUP BY columns]
[HAVING group-selection-condition]
[ORDER BY column-names || aliases || column-numbers];
```

- The SELECT clause is mandatory and carries out the relational project operation. The FROM clause is also mandatory. It identifies one or more tables and/or views from which to retrieve the column data displayed in a result table.
- The WHERE clause is optional and carries out the relational select operation. It specifies which rows are to be selected.
- The GROUP BY clause is optional. It organizes data into groups by one or more column names listed in the SELECT clause. The optional HAVING clause sets conditions regarding which groups to include in a result table. The groups are specified by the GROUP BY clause.
- The ORDER BY clause is optional. It sorts query results by one or more columns in ascending or descending order.

For Example,

```
SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID
  FROM CUSTOMERS
  WHERE SALARY > 4500)
```

6.4. FUNCTIONS

SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc.

A function is a database object in SQL Server. Basically, it is a set of SQL statements that accept only input parameters, perform actions and return the result. A function can return an only a single value or a table. We can't use a function to Insert, Update, Delete records in the database tables. Different Types of SQL Server & SQL Database Functions

System Defined Function

These functions are defined by SQL Server for a different purpose. We have two types of system defined function in SQL Server

1. Scalar Function

Scalar functions operates on a single value and returns a single value. Below is the list of some useful Sql Server Scalar functions.

- abs(-10.67)
  This returns absolute number of the given number means 10.67.
- rand(10)
  This will generate random number of 10 characters.
- round(17.56719,3)
  This will round off the given number to 3 places of decimal means 17.567
- upper('dotnet')
  This will returns upper case of given string means 'DOTNET'
- lower('DOTNET')
  This will returns lower case of given string means 'dotnet'
- ltrim(' dotnet')
  This will remove the spaces from left hand side of 'dotnet' string.
- convert(int, 15.56)
  This will convert the given float value to integer means 15.

2. Aggregate Functions

SQL Server aggregate functions perform a calculation on a set of values and return a single value. With the exception of the COUNT aggregate function, all other aggregate functions ignore NULL values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement.

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

User Defined Function

These functions are created by the user in the system database or in the user-defined database. There are three types of user-defined functions.

1. Scalar Function

User-defined scalar functions return a single data value of the type defined in the RETURNS clause. For an inline scalar function, there is no function body; the scalar value is the result of a single statement. For a multi-statement scalar function, the function body, defined in a BEGIN...END block contains a series of Transact-SQL statements that return the single value. The return type can be any data type except text, ntext, image, cursor, and timestamp.

2. Table-Valued Functions

User-defined table-valued functions return a table data type. For an inline table-valued function, there is no function body; the table is the result set of a single SELECT statement. Examples.

3. System Functions

SQL Server provides many system functions that you can use to perform a variety of operations. They cannot be modified.

6.5. GROUP BY FEATURE

The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

- "SELECT statements..." is the standard SQL SELECT command query.
- "GROUP BY column_name1" is the clause that performs the grouping based on column_name1.
- "[,column_name2,...]" is optional; represents other column names when the grouping is done on more than one column.
- "[HAVING condition]" is optional; it is used to restrict the rows affected by the GROUP BY clause. It is similar to the WHERE clause.

Example
Consider the CUSTOMERS table is having the following records −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

```
SELECT NAME, SUM(SALARY) FROM CUSTOMERS GROUP BY NAME;
```

- The GROUP BY Clause is used to group rows with same values .
- The GROUP BY Clause is used together with the SQL SELECT statement.
- The SELECT statement used in the GROUP BY clause can only be used contain column names, aggregate functions, constants and expressions.
- The HAVING clause is used to restrict the results returned by the GROUP BY clause.

6.6. UPDATING THE DATABASE

Information stored in a database is unlikely to remain unchanged. When the source of these changes is a dataset elsewhere, bulks updates must occur.

These updates may be as simple as reloading an entire set of data, totally replacing the existing content. In some cases the table is restructured before reloading, in other cases the table is truncated (emptied of its contents) but the structure left as before.

More complex updates involve processing individual records (leaving others untouched). In some cases the entire record is replaced, in other cases only individual fields of a record are updated.

The UPDATE statement changes existing data in one or more rows in a table. The following illustrates the syntax of the UPDATE statement:

```
UPDATE table
SET
   column1 = new_value1,
   column2 = new_value2,
   ...
WHERE
   Condition;
```

To update data in a table, you need to:

- First, specify the table name that you want to change data in the UPDATE clause.
- Second, assign a new value for the column that you want to update. In case you want to update data in multiple columns, each column = value pair is separated by a comma (,).
- Third, specify which rows you want to update in the WHERE clause. The WHERE clause is optional. If you omit the WHERE clause, all rows in the table will be updated.

The database engine issues a message specifying the number of affected rows after you execute the statement.

For Example,

```
UPDATE employees
SET
   lastname = 'Hill'
WHERE
   employeeID = 3;
```

6.7. DATA DEFINITION FACILITIES

A database language standard specifies the semantics of various components of a database management system (DBMS). In particular, it defines the structures and operations of a data model implemented by the DBMS, as well as other components that support data definition, data access, security, programming language interface, and data administration. The SQL

standard specifies data definition, data manipulation, and other associated facilities of a DBMS that supports the relational data model.

A database language standard is appropriate for all database applications where data will be shared with other applications, where the life of the application is longer than the life of current equipment, or where the application is to be understood and maintained by programmers other than the original ones.

The basic structure of the relational model is a table, consisting of rows and columns. Data definition includes declaring the name of each table to be included in a database, the names and data types of all columns of each table, constraints on the values in and among columns, and the granting of table manipulation privileges to prospective users. Tables can be accessed by inserting new rows, deleting or updating existing rows, or selecting rows that satisfy a given search condition for output. Tables can be manipulated to produce new tables by Cartesian products, unions, intersections, joins on matching columns, or projections on given columns.

The SQL data definitions for creation of a table begin after the command to create them, not in a separate screen panel or program. The SQL statement Create Table starts the processing command. It is going to create the table name specified after the word "Table." So following the Create Table, you specify the schema / collection in which the table will be created, immediately followed by the name of the table that is being created

----

6.8. LET US SUM UP

In this unit, you have learnt about the categories of SQL commands and basic concepts of predefined and user defined functions, and database updation. Thus, the SQL commands unit would have brought you to closer to know the concept of database access and its working principle.

6.9. UNIT – END QUESTIONS

1. Explain about the various categories of SQL Commands.
2. Write short notes on functions and GROUP BY feature of SQL.
3. How to updating the database using SQL.

6.10. ANSWER TO CHECK YOUR PROGRESS

1. SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an ANSI (American National Standards Institute) standard language, but there are many different versions of the SQL language. SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.
   When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task. There are various components included in this process. These components are – Query Dispatcher, Optimization Engines, Classic Query Engine, SQL Query Engine, etc.
   A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

2. SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. A function is a database object in SQL Server. Basically, it is a set of SQL statements that accept only input parameters, perform actions and return the result. A function can return an only a single value or a table. We can’t use a function to Insert, Update, Delete records in the database tables. Different Types of SQL Server & SQL Database Functions
   The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

3. Information stored in a database is unlikely to remain unchanged. When the source of these changes is a dataset elsewhere, bulks updates must occur.
   These updates may be as simple as reloading an entire set of data, totally replacing the existing content. In some cases the table is restructured before reloading, in other cases the table is truncated (emptied of its contents) but the structure left as before.

UNIT VII – SQL - II

Structure

7.1. INTRODUCTION

In this lesson you will be aware with the views. Embedded SQL*, transaction processing and ACID properties of DBMS. These SQL elements and properties include the embedded SQL statements and its variable creation and to study the ACID properties of database system like consistency and isolation and atomicity and durability of database management system.

7.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain embedded SQL* statements.
- Define transaction processing and variables and exceptions of the database system.
- Define ACID properties of the DBMS.

7.3. VIEWS

A database view is a virtual table or logical table which is defined as a SQL SELECT query with joins. Because a database view is similar to a database table, which consists of rows and columns, so you can query data against it. Most database management systems, including MySQL, allow you to detain the underlying tables through the database view with some prerequisites. A database view is dynamic because it is not related to the physical schema. The database system stores views as a SQL SELECT statement with joins. When the data of the tables changes, the view reflects that changes as well.

A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity. A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depend on the written SQL query to create a view.

Views, which are a type of virtual tables, allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows −

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

To include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

For Example,

```
CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS;
```

Updating a View

A view can be updated under certain conditions which are given below −

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view.

Views are used for a number of reasons

- Performance - since the joins already made and the tables are virtual, it's quicker to run a query on that dataset
- Security - users don't get access to the underlying tables and therefore can't mess up your data. The view controls what columns and tables are available to the user

---

7.4. EMBEDDED SQL*

Embedded SQL is the one which combines the high level language with the DB language like SQL. It allows the application languages to communicate with DB and get requested result. The high level languages which supports embedding SQLs within it are also known as host language. There are different host languages which support embedding SQL within it like C, C++, ADA, Pascal, FORTRAN, Java etc. When SQL is embedded within C or C++, then it is known as Pro*C/C++ or simply Pro*C language. Pro*C is the most commonly used embedded SQL. Let us discuss below embedded SQL with respect to C language.

When SQL is embedded within C language, the compiler processes the compilation in two steps. It first extracts all the SQL codes from the program and the pre-compiler will compile the SQL code for its syntax, correctness, execution path etc. Once pre-compilation is done, these executable codes are embedded into the C code. Then the C compiler will compile the code and execute the code. Thus the compilation takes place in two steps – one for SQL and one for application language. Hence these types of compilation require all the query, data value etc to be known at the compilation time itself to generate the executable code. Otherwise C or any high level language cannot compile the code. Hence the SQL codes written are static and these embedded SQL is also known as static SQL. Thus SQLs know how to access DB, which queries to execute, which values to be inserted/ deleted/updated etc.

Structure of embedded SQL defines step by step process of establishing a connection with DB and executing the code in the DB within the high level language.

Connection to DB

This is the first step while writing a query in high level languages. First connection to the DB that we are accessing needs to be established. This can be done using the keyword CONNECT. But it has to proceed with 'EXEC SQL' to indicate that it is a SQL statement.

Declaration Section

Once connection is established with DB, we can perform DB transactions. Since these DB transactions are dependent on the values and variables of the host language. Depending on their values, query will be written and executed. Similarly, results of DB query will be returned to the host language which will be captured by the variables of host language. Hence we need to declare the variables to pass the value to the query and get the values from query. There are two types of variables used in the host language.

- Host variable: These are the variables of host language used to pass the value to the query as well as to capture the values returned by the query. Since SQL is dependent on host language we have to use variables of host language and such variables are known as host variable. But these host variables should be declared within the SQL area or within SQL code. That means compiler should be able to differentiate it from normal C variables. Hence we have to declare host variables within BEGIN DECLARE and END DECLARE section. Again, these declare block should be enclosed within EXEC SQL and ';'.
- Indicator Variable: These variables are also host variables but are of 2 byte short type always. These variables are used to capture the NULL values that a query returns or to INSERT/ UPDATE any NULL values to the tables. When it is used in a SELECT query, it captures any NULL value returned for any column.

Execution Section

This is the execution section, and it contains all the SQL queries and statements prefixed by 'EXEC SQL'.

7.5. DECLARING VARIABLES AND EXCEPTIONS

Exceptions we learnt that there are three ways of declaring user-define exceptions in Oracle Database. Declaring a user-define exception using Exception variable is a three step process. These three steps are –

1. Declare a variable of exception datatype – This variable is going to take the entire burden on its shoulders.
2. Raise the Exception – This is the part where you tell the compiler about the condition which will trigger the exception.
3. Handle the exception – This is the last section where you specify what will happen when the error which you raised will trigger.

Step 1: Declare a variable of Exception datatype

SQL variable you can declare an Exception variable in declaration section of the anonymous as well as named PL/SQL block. This exception variable will then work as user-define

```
DECLARE
 var_dividend NUMBER := 24;
 var_divisor NUMBER := 0;
 var_result NUMBER;
 ex_DivZero EXCEPTION;
```

In this declaration section we have 4 variables. Among these 4 variables first 3 are normal Number datatype variables and the 4th one which is ex_DivZero is the special EXCEPTION datatype variable. This variable will become our User-Define Exception for this program.

Step 2: Raise the Exception

The next step after declaring an Exception variable is to raise the exception. To raise the exception in SQL we use Raise statement.

Raise statement is a special kind of SQL statement which changes the normal flow of execution of the code. As soon as compiler comes across a raise condition it transfers the control over to the exception handler.

```
BEGIN
 IF var_divisor = 0 THEN
  RAISE ex_DivZero;
 END IF
```

Here raise condition is accompanied with the IF-THEN condition. With the help of this we can avoid unwanted switches during the control flow of the program. Using If Condition we are making sure that this error will come into action only when the divisor is equal to 0.

Step 3: Handle the exception

That is the main section of the code. Here we write the logic for our user-define exception and tell the compiler what it should do if and when that error occurs.

```
EXCEPTION WHEN ex_DivZero THEN
   DBMS_OUTPUT.PUT_LINE('Error Error - Your Divisor is Zero');
END;
/
```

Here we have the exception handler for the variable ex_DivZero. In the exception handling section we have a DBMS OUTPUT statement which will get displayed when our user define error which is ex_DivZero occurs.

7.6. EMBEDDING SQL STATEMENTS

Structured Query Language (SQL) is a standardized language that you can use to manipulate database objects and the data that they contain. Despite differences between host languages, embedded SQL applications are made up of three main elements that are required to setup and issue an SQL statement.

The elements you must create when you write an embedded SQL application include:

- A DECLARE SECTION for declaring host variables. The declaration of the SQLCA structure does not need to be in the DECLARE section.
- The main body of the application, which consists of the setup and execution of SQL statements.
- Placements of logic that either commit or rollback the changes made by the SQL statements.

The structured query language provides us 2 features:

- It allows us to write queries.
- It allows us to use it in programming languages so that database can be accessed through application programs also.

There are many queries that are expressed in programming languages like C, C++, Java but can’t be expressed in SQL. For writing such queries we need to embed the SQL in general purpose programming languages. The mixture of SQL and general purpose programming languages is called embedded SQL.

There are some special embedded SQL statements which are used to retrieve the data into the program. There is a special SQL precompiler that accepts the combined source code with other programming tools and converts them into an executable program.

Concepts for Embedding the SQL Statements

We can mix the SQL statements directly into general purpose programming language like C, Java or Pascal. There are some techniques to embed SQL statements in the programming languages.

- The programming language in which the SQL statements are embedded is called the host language. The SQL statements and host language statements make the source program which is fed to a SQL precompiler for processing the SQL statements.
- The host programming languages variables can be referenced in the embedded SQL statements, which allows values calculated by the programs to be used by SQL statements.
- There are some special program variables which are used to assign null values to database columns. These program variables support the retrieval of null values from the database.

Embedded SQL Program Development

Since the embedded SQL is a mixture of SQL and programming language, so it cannot be fed directly to a general purpose programming language compiler. Actually the program execution is a multi-step which is as follows.

- First, the embedded SQL source code is fed to the SQL precompiler. The precompiler scans the program and processes the embedded SQL statements present in the code. There can be different precompiler for different type of programming languages.
- After processing the source code, the precompiler produces 2 files as its output. The first file contains the source program without embedded SQL statements and the second file contains all the embedded SQL statements used in the program.
- The first file produced by precompiler (that contains the source program) is fed to the compiler for the host programming language (like C compiler). The compiler processes the source code and produces object code as its output.
- Now the linker takes the object modules produced by the compiler and link them with various library routines and produces an executable program.

- The database request modules, produced by the precompiler (in steps) are submitted to a special BIND program. The BIND program examines the SQL statements, parse them, validates them, optimizes them and finally produces an application plan for each statement. The result is a combined application plan for the entire program, that represents a DBMS-executable version of its embedded SQL statements. The BIND program stores the plan in the database, usually assigning it the name of the application program that has created it.

7.7. TRANSACTION PROCESSING

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on that table. It is important to control these transactions to ensure the data integrity and to handle database errors.

Transaction processing is used to maintain database integrity by ensuring that batches of SQL operations execute completely or not at all.

When working with transactions and transaction processing, there are a few keywords that'll keep reappearing. Here are the terms you need to know:

- Transaction - A block of SQL statements
- Rollback - The process of undoing specified SQL statements
- Commit - Writing unsaved SQL statements to the database tables
- Savepoint - A temporary placeholder in a transaction set to which you can issue a rollback (as opposed to rolling back an entire transaction)

Properties of Transactions

Transactions have the following four standard properties, usually referred to by the acronym **ACID**.

- **Atomicity**.
- **Consistency**.
- **Isolation**.
- **Durability** .

Transaction Control

The following commands are used to control transactions.

- **COMMIT** − to save the changes.
- **ROLLBACK** − to roll back the changes.
- **SAVEPOINT** − creates points within the groups of transactions in which to ROLLBACK.
- **SET TRANSACTION** − Places a name on a transaction.

Transactional Control Commands

Transactional control commands are only used with the **DML Commands** such as - INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

The COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for the COMMIT command is as follows.

COMMIT;

The ROLLBACK Command

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows −

ROLLBACK;

7.8. CONSISTENCY AND ISOLATION

In database systems, **ACID** (Atomicity, Consistency, Isolation, and Durability) refers to a standard set of properties that guarantee database transactions are processed reliably.

ACID is especially concerned with how a database recovers from any failure that might occur while processing a transaction.

An ACID-compliant DBMS ensures that the data in the database remains accurate and consistent despite any such failures.

Consistency

Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination

thereof. This prevents database corruption by an illegal transaction, but does not guarantee that a transaction is correct. Referential integrity guarantees the primary key - foreign key relationship. Consistency is the ACID property that ensures that any changes to values in an instance are consistent with changes to other values in the same instance. A consistency constraint is a predicate on data which serves as a precondition, post-condition, and transformation condition on any transaction.

This SQL ACID property ensures the database consistency. It means, whatever happens in the middle of the transaction, this property will never leave your database in half-completed state.

- If the transaction is completed successfully then it will apply all the changes to the database.
- If there is any error in a transaction then all the changes that are already made will be rolled back automatically. It means, database will be restored to its state that it had before the transaction started.
- If there is a system failure in the middle of the transaction then also, all the changes that are already made will automatically roll back.

Isolation

This is easily the most often misunderstood of the ACID transaction properties.

In principle, a completely isolated transaction executes as the only task executing against the database during its lifetime. Other transactions can only start once the current transaction has completely finished (i.e. committed or rolled back). Executed this way, a transaction would truly be an atomic operation, in the strict sense that a non-database person would ascribe to the phrase.

In practice, database transactions operate instead with a degree of isolation specified by the currently effective transaction isolation level (which applies equally to stand-alone statements, remember). This compromise (the degree of isolation) is the practical consequence of the trade-offs between concurrency and correctness mentioned earlier. A system that literally processed transactions one-by-one, with no overlap in time, would provide complete isolation but overall system throughput would likely be poor.

7.9. AUTOMICITY AND DURABILITY

Atomicity

Transactions are often composed of multiple statements. Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely: if any of the statements constituting a transaction fails to complete, the entire transaction fails and

the database is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes.

A series of database operations in an atomic transaction will either all occur, or none will occur. The series of operations cannot be separated with only some of them being executed, which makes the series of operations "indivisible". A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright. In other words, atomicity means indivisibility and irreducibility. Alternatively, we may say that a Logical transaction may be made of, or composed of, one or more (several), Physical transactions. Unless and until all component Physical transactions are executed, the Logical transaction will not have occurred – to the effects of the database.

Durability

Once the transaction is successfully completed, then the changes it has made to the database will be permanent. Even if there is a system failure, or any abnormal changes also, this SQL acid property will safeguard the committed data.

7.10. LET US SUM UP

In this unit, you have learnt about the embedded SQL* and its statements, variable declarations, transaction processing and ACID properties of the DBMS. Thus, the SQL statements-II unit would have brought you to closer to know the concept of embedded SQL statements and implementation of ACID properties like isolation, consistency, atomicity and durability and its working principle.

7.11. UNIT – END QUESTIONS

1. Identify the use of embedded SQL* statements and its variables.
2. Discuss about the ACID properties of DBMS.

7.12. ANSWER TO CHECK YOUR PROGRESS

1. Embedded SQL is the one which combines the high level language with the DB language like SQL. It allows the application languages to communicate with DB and get requested result. The high level languages which supports embedding SQLs within it are also known as host language. Structured Query Language (SQL) is a standardized language that you can use

to manipulate database objects and the data that they contain. Despite differences between host languages, embedded SQL applications are made up of three main elements that are required to setup and issue an SQL statement.

2. In database systems, **ACID** (Atomicity, Consistency, Isolation, and Durability) refers to a standard set of properties that guarantee database transactions are processed reliably. ACID is especially concerned with how a database recovers from any failure that might occur while processing a transaction.

   An ACID-compliant DBMS ensures that the data in the database remains accurate and consistent despite any such failures.

   Consistency - Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof.

   Isolation - This is easily the most often misunderstood of the ACID transaction properties. In principle, a completely isolated transaction executes as the only task executing against the database during its lifetime. Other transactions can only start once the current transaction has completely finished

   Atomicity - Transactions are often composed of multiple statements. Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely: if any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes.

   Durability - Once the transaction is successfully completed, then the changes it has made to the database will be permanent. Even if there is a system failure, or any abnormal changes also, this SQL acid property will safeguard the committed data.

UNIT VIII – RELATIONAL ALGEBRA

Structure

8.1. INTRODUCTION

In this lesson you will be aware with the relational algebra and is operations. These basic relational algebra operations include union, difference, intersection and Cartesian product. Apart from the basic operations it consist of some additional operations used like projection, selection, JOIN and division.

8.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain basic operations of relational algebra.
- Define additional relational algebra operations used..

8.3. BASIC OPERATION

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages − relational algebra and relational calculus.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Relational algebra is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action.

Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations. We have divided these operations in two categories:

1. Basic Operations.
2. Derived Operations.

8.3.1. Union (U)

Union operator is denoted by ∪ symbol and it is used to select all the rows (tuples) from two tables (relations).

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold -

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.
- Example
- Consider the following tables.

Table A			Table B	
column 1	column 2		column 1	column 2
1	1		1	1
1	2		1	3

- A ∪ B gives

Table A ∪ B	
column 1	column 2
1	1
1	2
1	3

8.3.2. Difference (-)

This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B. Example-

Consider the following two relations R and S-

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation R

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation S

Then, R – S is-

ID	Name	Subject
200	Pooja	Maths
300	Komal	Science

Relation R – S

8.3.3. Intersection

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Let's consider two relations R1 and R2 both have same columns and we want to select all those tuples (rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

Only those rows that are present in both the tables will appear in the result set.

An intersection is defined by the symbol ∩

A ∩ B

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.

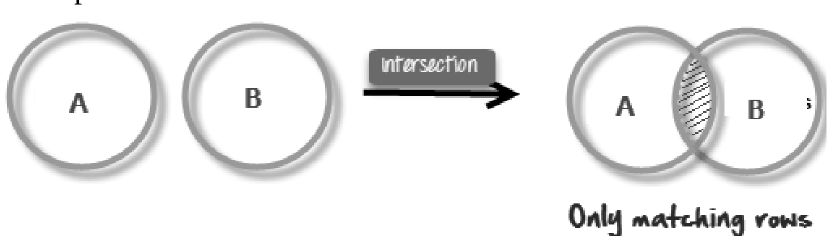


Figure 14: -Intersection Operation

Example:
A ∩ B

Table A ∩ B	
column 1	column 2
1	1

8.3.4. Cartesian product (x)

This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

Example – Cartesian product

$\sigma_{\text{column 2} = '1'}$ (A X B)

Output – The above example shows all rows from relation A and B whose column 2 has value 1

σ column 2 = '1' (A X B)	
column 1	column 2
1	1
1	1

It performs the function of combining information from two or more relations into one. It represented by r X s, where r and s are relations.

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

8.4. ADDITIONAL RELATIONAL ALGEBRA OPERATIONS

While the fundamental relational algebra operations — select, project, union, setdifference, cartesian-product, rename — form the basis for a broad spectrum of relational database queries and manipulations, it remains possible to define other operations over relations. Some operations are just notational shortcuts — they are completely rewritable in terms of the 6 fundamentals.

On the other hand, the "classical" relational algebra itself has been extended with new operations, with new meanings and manipulations. The basic relational-algebra operations have been extended in several ways. A simple extension is to allow arithmetic operations as part of projection.

8.4.1. Projection

Projection is possible as user's choice means the projection column may differ from table column order. Symbol for column query in projection **'π'**

The generalized-projection operation extends the fundamental projection operation by allowing arithmetic (or, in the most general case, overall transformative) functions in the projections attribute list. It is still denoted with Π, but now the straight-up attribute list A has changed into an expression list F_1, F_2, . . . , Fn:

$$\Pi F_1, F_2, \ldots, Fn\ (E)$$

E is any relational algebra expression, which is of course a relation. Fk may be any expression involving constant values and the attributes of E's resultant relation schema.

If we are interested in only certain attributes of a relation, we use the PROJECT operation to project the relation over these attributes only.

Therefore, the result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations:

one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns.

For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

πLname, Fname, Salary(EMPLOYEE)

The general form of the PROJECT operation is π<attribute list>(R)

Where π (pi) is the symbol used to represent the PROJECT operation, and is the desired sublist of attributes from the attributes of relation R.

Again, notice that R is, in general, a relational algebra expression whose result is a relation, which in the simplest case is just the name of a database relation.

The result of the PROJECT operation has only the attributes specified in <attribute list> in the same order as they appear in the list.
Hence, its degree is equal to the number of attributes in <attribute list>.

If the attribute list includes only nonkey attributes of R, duplicate tuples are likely to occur.

The PROJECT operation contains a key, removes any duplicate tuples, so the result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as duplicate elimination.

Duplicate elimination involves sorting or some other technique to detect duplicates and thus adds more processing.

If duplicates are not eliminated, the result would be a multiset or bag of tuples rather than a set. This was not permitted in the formal relational model, but is allowed in SQL.

8.4.2. Selection

The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition. One can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition.

$$\sigma_p(r)$$

σ is the predicate

r stands for relation which is the name of the table

p is prepositional logic

Alternatively, we can consider the SELECT operation to restrict the tuples in a relation to only those tuples that satisfy the condition. The SELECT operation can also be visualized as a horizontal partition of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.

σ <selection condition> (R)

Where the symbol σ (sigma) is used to denote the SELECT operator and the selection condition is a Boolean expression (condition) specified on the attributes of relation R.

Notice that R is generally a relational algebra expression whose result is a relation—the simplest such expression is just the name of a database relation. The relation resulting from the SELECT operation has the same attributes as R.

The SELECT operator is unary; that is, it is applied to a single relation. Moreover, the selection operation is applied to each tuple individually; hence, selection conditions cannot involve more than one tuple. The degree of the relation resulting from a SELECT operation—its number of attributes—is the same as the degree of R.

8.4.3. JOIN

Join is the most powerful operation is relational algebra. It allows us to retrieve data from multiple tables by establishing a relationship between them. Join is a projection of selection of product of two tables.

Type of joins

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

<u>INNER JOIN</u>

1. Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol θ

Example

$A \bowtie_{\theta} B$

Theta join can use any conditions in the selection criteria.

For example:

$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$

2. EQUI join:

When a theta join uses only equivalence condition, it becomes a equi join.

For example:

$A \bowtie_{A.column\ 2 = B.column\ 2} (B)$

A ⋈ A.column 2 = B.column 2 (B)	
column 1	column 2
1	1

EQUI join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

3. Natural Join

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example
Consider the following two tables

C	
Num	Square
2	4
3	9

D	
Num	Cube
2	8
3	18

C ⋈ D

C ⋈ D		
Num	Square	Cube
2	4	4
3	9	9

OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

1. Left Outer Join(A ⟕ B)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

2. Right Outer Join: (A ⟖ B)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

3. Full Outer Join: (A ⟗ B)

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

8.4.4. Division

The division is similarly performing as same as division set operation in mathematics between tuple (row) and attribute (column).

Here, is **some example is shown below** related to these operations take a look.

Table (A1) :

E. No.	E. Name	Salary	Dept. No.
1	A	5000	10
2	B	4000	10

Table (A2) :

Dept. No.	D. Name	Location
10	IT	Mumbai
20	Sales	Delhi
30	HR	Hyderabad

Division:

A1 / A2:

E. No	E. Name	Salary	D. Name	Location
1	A	5000	IT	Mumbai
1	A	5000	Sales	Delhi
1	A	5000	HR	Hyderabad
2	B	4000	IT	Mumbai
2	B	4000	Sales	Delhi
2	B	4000	HR	Mumbai

8.5. LET US SUM UP

In this unit, you have learnt about the relational algebra and its basic and additional operations. Thus, the relational algebra unit would have brought you to closer to know the concept of relational algebra basic operations like union, difference, intersection and Cartesian product and additional operations are projection, selection, JOIN and division are effectively implemented in DBMS.

8.6. UNIT – END QUESTIONS

1. List out the basic operations of relational algebra.
2. List out the additional operations of relational algebra.

8.7. ANSWER TO CHECK YOUR PROGRESS

1. Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

   Relational algebra is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action.

   Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations. We have divided these operations in two categories:

   - Basic Operations.
   - Derived Operations.

   Union operator is denoted by ∪ symbol and it is used to select all the rows (tuples) from two tables (relations).

   Difference (-) - This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Cartesian product (x) - This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone.

2. While the fundamental relational algebra operations — select, project, union, set difference, Cartesian-product, rename — form the basis for a broad spectrum of relational database queries and manipulations, it remains possible to define other operations over relations. Some operations are just notational shortcuts — they are completely rewritable in terms of the 6 fundamentals.

   On the other hand, the "classical" relational algebra itself has been extended with new operations, with new meanings and manipulations. The basic relational-algebra operations have been extended in several ways. A simple extension is to allow arithmetic operations as part of projection.

   Projection is possible as user's choice means the projection column may differ from table column order. Symbol for column query in projection **'π'.** The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition. One can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition. Join is the most powerful operation is relational algebra. It allows us to retrieve data from multiple tables by establishing a relationship between them. Join is a projection of selection of product of two tables. The division is similarly performing as same as division set operation in mathematics between tuple (row) and attribute (column).

BLOCK III: - E-PAYMENT SYSTEMS AND CRM, SCM

UNIT IX – RELATIONAL CALCULUS

Structure

UNIT IX – RELATIONAL CALCULUS

9.1. INTRODUCTION

As you know by now, how to use the relational calculus and find the comparison of relational algebra and relational calculus for database management system. The queries are playing a vital role in database system and database management. In this lesson, you have learnt about the types of relational calculus and difference between the relational algebra and relational calculus.

9.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain tuple relational calculus and its queries.
- Explain domain relational calculus.

9.3. TUPLE RELATIONAL CALCULUS

Relational calculus is a query language which is non-procedural, and instead of algebra, it uses mathematical predicate calculus. The relational calculus is not the same as that of differential and integral calculus in mathematics but takes its name from a branch of symbolic logic termed as predicate calculus.

Relational calculus is a non procedural query language. It uses mathematical predicate calculus instead of algebra. It provides the description about the query to get the result where as relational algebra gives the method to get the result. It informs the system what to do with the relation, but does not inform how to perform it.

For example, steps involved in listing all the students who attend 'Database' Course in relational algebra would be

- SELECT the tuples from COURSE relation with COURSE_NAME = 'DATABASE'
- PROJECT the COURSE_ID from above result
- SELECT the tuples from STUDENT relation with COUSE_ID resulted above.

In the case of relational calculus, it is described as below:

Get all the details of the students such that each student have course as 'Database'.

See the difference between relational algebra and relational calculus here. From the first one, we are clear on how to query and which relations to be queried. But the second tells what needs to be done to get the students with 'database' course. But it does tell us how we need to proceed to achieve this. Relational calculus is just the explanative way of telling the query.

There are two types of relational calculus –

- Tuple Relational Calculus (TRC) and
- Domain Relational Calculus (DRC).

9.3.1. Semantics of TRC Queries

A tuple relational calculus is a non procedural query language which specifies to select the tuples in a relation. It can select the tuples with range of values or tuples for certain attribute values etc. The resulting relation can have one or more tuples. It is denoted as below:

{t | P (t)} or **{t | condition (t)}** -- this is also known as expression of relational calculus

Where t is the resulting tuples, P(t) is the condition used to fetch t.

{t | EMPLOYEE (t) and t.SALARY>10000} - implies that it selects the tuples from EMPLOYEE relation such that resulting employee tuples will have salary greater than 10000. It is example of selecting a range of values.

{t | EMPLOYEE (t) AND t.DEPT_ID = 10} – this select all the tuples of employee name who work for Department 10.

The variable which is used in the condition is called tuple variable. In above example t.SALARY and t.DEPT_ID are tuple variables. In the first example above, we have specified the condition t.SALARY >10000. What is the meaning of it? For all the SALARY>10000, display the employees. Here the SALARY is called as bound variable. Any tuple variable with '*For All'* (?) or '*there exists'* (?) condition is called bound variable. Here, for any range of values of SALARY greater than 10000, the meaning of the condition remains the same. Bound variables are those ranges of tuple variables whose meaning will not change if the tuple variable is replaced by another tuple variable.

In the second example, we have used DEPT_ID= 10. That means only for DEPT_ID = 10 display employee details. Such variable is called free variable. Any tuple variable without any 'For All' or 'there exists' condition is called Free Variable. If we change DEPT_ID in this condition to some other variable, say EMP_ID, the meaning of the query changes. For example, if we change EMP_ID = 10, then above it will result in different result set. Free variables are those ranges of tuple variables whose meaning will change if the tuple variable is replaced by another tuple variable.

All the conditions used in the tuple expression are called as well formed formula – WFF. All the conditions in the expression are combined by using logical operators like AND, OR and NOT, and qualifiers like 'For All' (?) or 'there exists' (?). If the tuple variables are all bound variables in a WFF is called closed WFF. In an open WFF, we will have at least one free variable.

9.3.2. Examples of TRC Queries

Tuple relational calculus is used for selecting those tuples that satisfy the given condition.

Table: Student

First_Name	Last_Name	Age
----------	---------------	----
Ajeet	**Singh**	**30**
Chaitanya	**Singh**	**31**
Rajeev	**Bhatia**	**27**
Carl	**Pratap**	**28**

Let's write relational calculus queries.

Query to display the last name of those students where age is greater than 30

{ t.Last_Name | Student(t) AND t.age > 30 }

In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part we specify the fields which we want to display for the selected tuples.

The result of the above query would be:

Last_Name
Singh

Query to display all the details of students where Last name is ‘Singh’

{ t | Student(t) AND t.Last_Name = 'Singh' }

Output:

First_Name	**Last_Name**	**Age**
Ajeet	**Singh**	**30**
Chaitanya	**Singh**	**31**

9.4. DOMAIN RELATIONAL CALCULUS

The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes.

Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ¬ (not). It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

In contrast to tuple relational calculus, domain relational calculus uses list of attribute to be selected from the relation based on the condition. It is same as TRC, but differs by selecting the attributes rather than selecting whole tuples. It is denoted as below:

$$\{< a_1, a_2, a_3, \dots a_n > | P(a_1, a_2, a_3, \dots a_n)\}$$

Where $a_1, a_2, a_3, \dots a_n$ are attributes of the relation and P is the condition.

For example, select EMP_ID and EMP_NAME of employees who work for department 10

{<EMP_ID, EMP_NAME> | <EMP_ID, EMP_NAME> ? EMPLOYEE Λ DEPT_ID = 10}

Get name of the department name that Alex works for.

{DEPT_NAME |< DEPT_NAME > ? DEPT Λ ? DEPT_ID (<DEPT_ID> ? EMPLOYEE Λ EMP_NAME = Alex)}

Here green color expression is evaluated to get the department Id of Alex and then it is used to get the department name form DEPT relation.

Let us consider another example where select EMP_ID, EMP_NAME and ADDRESS the employees from the department where Alex works. What will be done here?

{<EMP_ID, EMP_NAME, ADDRESS, DEPT_ID > | <EMP_ID, EMP_NAME, ADDRESS, DEPT_ID> ? EMPLOYEE Λ ? DEPT_ID (<DEPT_ID> ? EMPLOYEE Λ EMP_NAME = Alex)}

First, formula is evaluated to get the department ID of Alex (green color), and then all the employees with that department is searched (red color).

9.5. RELATIONAL ALGEBRA VS RELATIONAL CALCULUS

Relational Algebra and Relational Calculus are the formal query languages for a relational model. Both form the base for the SQL language which is used in most of the relational DBMSs. Relational Algebra is a procedural language. On the other hands, Relational Calculus is a declarative language.

Comparison Chart: -

BASIS FOR COMPARISON	RELATIONAL ALGEBRA	RELATIONAL CALCULUS
Basic	Relational Algebra is a Procedural language.	Relational Calculus is Declarative language.
States	Relational Algebra states how to obtain the result.	Relational Calculus states what result we have to obtain.
Order	Relational Algebra describes the order in which operations have to be performed.	Relational Calculus does not specify the order of operations.
Domain	Relational Algebra is not domain dependent.	Relation Claculus can be domain dependent.
Related	It is close to a programming language.	It is close to the natural language.

Key Differences between Relational Algebra and Relational Calculus

1. The basic difference between Relational Algebra and Relational Calculus is that Relational Algebra is a Procedural language whereas, the Relational Calculus is a Non-Procedural, instead it is a Declarative language.
2. The Relational Algebra defines how to obtain the result whereas, the Relational Calculus define what information the result must contain.
3. Relational Algebra specifies the sequence in which operations have to be performed in the query. On the other hands, Relational calculus does not specify the sequence of operations to perform in the query.

4. The Relational Algebra is not domain dependent whereas, the Relational Calculus can be domain dependent as we have Domain Relational Calculus.
5. The Relational Algebra query language is closely related to programming language whereas; the Relational Calculus is closely related to the Natural Language.

9.5. LET US SUM UP

In this unit, you have learnt about the relational calculus and its types, each type having a predefined query system. Thus, the relational calculus unit would have brought you to closer to know the concept of relational calculus and types of relational calculus and queries used in each type is identified.

9.6. UNIT – END QUESTIONS

1. Describe about the tuple relational calculus and its queries.
2. Describe about the domain relational calculus with example.

9.7. ANSWER TO CHECK YOUR PROGRESS

1. Relational calculus is just the explanative way of telling the query. There are two types of relational calculus –
    1. Tuple Relational Calculus (TRC) and
    2. Domain Relational Calculus (DRC).

   A tuple relational calculus is a non procedural query language which specifies to select the tuples in a relation. It can select the tuples with range of values or tuples for certain attribute values etc. The resulting relation can have one or more tuples.

2. The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes.

   Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives $\wedge$ (and), $\vee$ (or) and $\neg$ (not). It uses Existential ($\exists$) and Universal Quantifiers ($\forall$) to bind the variable.

In contrast to tuple relational calculus, domain relational calculus uses list of attribute to be selected from the relation based on the condition. It is same as TRC, but differs by selecting the attributes rather than selecting whole tuples.

UNIT X – NORMALIZATION

Structure

UNIT X – NORMALIZATION

10.1. INTRODUCTION

In this lesson you will be know about the concept of normalization and its types. This normalization concept elaborates the anomalies of a database, properties of normalized relations and various types of normalization techniques. Apart from the normalization technique the functional dependencies are identified in this unit.

10.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain the functional dependencies of the database.
- Define the anomalies of a database.
- Explain the normalization and its techniques.

10.3. FUNCTIONAL DEPENDENCY

Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other. Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs.

To understand the concept thoroughly, let us consider P is a relation with attributes A and B. Functional Dependency is represented by -> (arrow sign)

Then the following will represent the functional dependency between attributes with an arrow sign:

A -> B

Rules of Functional Dependencies

Below given are the three most important rules for Functional Dependency:

- Reflexive rule –. If X is a set of attributes and Y is_subset_of X, then X holds a value of Y.
- Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies.
- Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y.

Types of Functional Dependencies

- Multivalued dependency:
- Trivial functional dependency:
- Non-trivial functional dependency:
- Transitive dependency:

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation.

The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.

So, X -> Y is a trivial functional dependency if Y is a subset of X.

Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

A transitive is a type of functional dependency which happens when t is indirectly formed by two functional dependencies.

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

---

10.4. ANOMALIES IN A DATABASE

Database anomalies are the problems in relations that occur due to redundancy in the relations. These anomalies affect the process of inserting, deleting and modifying data in the relations. Some important data may be lost if a relations is updated that contains database anomalies. It is important to remove these anomalies in order to perform different processing on the relations without any problem.

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

- **Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.
- **Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.
- **Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data.

10.5. PROPERTIES OF NORMALIZED RELATIONS

Normalization is the process of removing redundant data from your tables in order to improve storage efficiency, data integrity and scalability. This improvement is balanced against an increase in complexity and potential performance losses from the joining of the normalized tables at query-time. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

Why We Need Normalization?

Normalization is the aim of well design Relational Database Management System (RDBMS). It is step by step set of rules by which data is put in its simplest forms. We normalize the relational database management system because of the following reasons:

- Minimize data redundancy i.e. no unnecessarily duplication of data.
- To make database structure flexible i.e. it should be possible to add new data values and rows without reorganizing the database structure.
- Data should be consistent throughout the database i.e. it should not suffer from following anomalies.
- Insert Anomaly - Due to lack of data i.e., all the data available for insertion such that null values in keys should be avoided. This kind of anomaly can seriously damage a database.
- Update Anomaly - It is due to data redundancy i.e. multiple occurrences of same values in a column. This can lead to inefficiency.
- Deletion Anomaly - It leads to loss of data for rows that are not stored elsewhere. It could result in loss of vital data.

- Complex queries required by the user should be easy to handle.

Benefits of Normalization

There are many benefits of normalizing a database. Here are some of the key benefits:

- Minimizes data redundancy (duplicate data).
- Minimizes null values.
- Results in a more compact database (due to less data redundancy/null values).
- Minimizes/avoids data modification issues.
- Simplifies queries.
- The database structure is cleaner and easier to understand. You can learn a lot about a relational database just by looking at its schema.
- You can extend the database without necessarily impacting the existing data.
- Searching, sorting, and creating indexes can be faster, since tables are narrower, and more rows fit on a data page.

There are several stages of the normalization process. These are called the *first normal form (1NF), the second normal form (2NF), the third normal form (3.NF), Boyce-Codd normal form (BCNF), the fourth normal form (4NF) and the fifth normal form* (5NF).

10.6. FIRST NORMALIZATION

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

If a table is not properly normalized and has data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized.

The first normal form expects you to follow a few simple rules while designing your database, and they are:

Rule 1: Single Valued Attributes

Each column of your table should be single valued which means they should not contain multiple values.

Rule 2: Attribute Domain should not change

This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

Rule 3: Unique name for Attributes/Columns

This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data. If one or more columns have same name, then the DBMS system will be left confused.

Rule 4: Order doesn't matters

This rule says that the order in which you store the data in your table doesn't matter. Here is our table, with some sample data added to it.

Roll_no	Name	Subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

Our table already satisfies 3 rules out of the 4 rules, as all our column names are unique, we have stored data in the order we wanted to and we have not inter-mixed different type of data in columns.

But out of the 3 different students in our table, 2 have opted for more than 1 subject. And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.

Here is our updated table and it now satisfies the First Normal Form.

Roll_no	Name	Subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

By doing so, although a few values are getting repeated but values for the subjectcolumn are now atomic for each record/row.

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

10.7. SECOND NORMAL FORM

For a table to be in the Second Normal Form, it must satisfy two conditions:

1. The table should be in the First Normal Form.
2. There should be no Partial Dependency.

For a table to be in the Second Normal form it should be in the First Normal form and it should not have Partial Dependency.

Partial Dependency exists, when for a composite primary key, any attribute in the table depends only on a part of the primary key and not on the complete primary key.

To remove Partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.

The entity should be considered already in 1NF, and all attributes within the entity should depend solely on the unique identifier of the entity.

Example:

Sample Products table:

productID	product	Brand
1	Monitor	Apple
2	Monitor	Samsung
3	Scanner	HP
4	Head phone	JBL

Product table following 2NF:

Products Category table:

productID	product
1	Monitor
2	Scanner
3	Head phone

Brand table:

brandID	brand
1	Apple
2	Samsung
3	HP
4	JBL

Products Brand table:

pbID	productID	brandID
1	1	1
2	1	2
3	2	3
4	3	4

10.8. THIRD NORMAL FORM

For a table to be in the third normal form,

1. It should be in the Second Normal form.
2. And it should not have Transitive Dependency.

Transitive Dependency?

With exam_name and total_marks added to our Score table, it saves more data now. Primary key for our Score table is a composite key, which means it's made up of two attributes or columns → student_id + subject_id.

Our new column exam_name depends on both student and subject. For example, a mechanical engineering student will have Workshop exam but a computer science student won't. And for some subjects you have Prctical exams and for some you don't. So we can say that exam_name is dependent on both student_id and subject_id.

The column total_marks depends on exam_name as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.

But, exam_name is just another column in the score table. It is not a primary key or even a part of the primary key, and total_marks depends on it.

This is Transitive Dependency. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.

How to remove Transitive Dependency?

Again the solution is very simple. Take out the columns exam_name and total_marksfrom Score table and put them in an Exam table and use the exam_id wherever required.

Score Table: In 3rd Normal Form

score_id	student_id	subject_id	marks	exam_id

The new Exam table

exam_id	exam_name	total_marks
1	Workshop	200
2	Mains	70
3	Practicals	30

Advantage of removing Transitive Dependency

The advantage of removing transitive dependency is,

- Amount of data duplication is reduced.
- Data integrity achieved.

10.9. BOYCE-CODD NORMAL FORM

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Below we have a college enrolment table with columns student_id, subject and professor.

student_id	**subject**	**professor**
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

In the table above student_id, subject together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the **2nd Normal Form** as their is no **Partial Dependency**.

And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.

But this table is not in **Boyce-Codd Normal Form**.

To make this relation(table) satisfy BCNF, we will decompose this table into two tables, **student** table and **professor** table.

Below we have the structure for both the tables.

Student Table

student_id	p_id
101	1
101	2

And, Professor Table

p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++
and so on...		

And now, this relation satisfies Boyce-Codd Normal Form.

10.10. FOURTH AND FIFTH NORMAL FORM

Fourth Normal Form

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the **Boyce-Codd Normal Form**.
2. And, the table should not have any **Multi-valued Dependency**.

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multi-valued dependent columns are moved to separate tables.

If you design your database carefully, you can easily avoid these issues.

Fifth Normal Form

A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

10.11. LET US SUM UP

In this unit, you have learnt about the need of normalization in the database, the concept of normalization, types of normalization. Thus, the normalization unit would have brought you to closer to know the concept of functional dependencies, anomalies of a database, normalization, and types of normalization.

10.12. UNIT – END QUESTIONS

1. Explain about the functional dependencies in the database.
2. Discuss about the anomalies of a database.
3. Define normalization. Explain the various types of normalization.

10.13. ANSWER TO CHECK YOUR PROGRESS

1. Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other. Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs.
   To understand the concept thoroughly, let us consider P is a relation with attributes A and B. Functional Dependency is represented by -> (arrow sign)
   Types of Functional Dependencies
   - Multivalued dependency:
   - Trivial functional dependency:
   - Non-trivial functional dependency:
   - Transitive dependency:

2. Database anomalies are the problems in relations that occur due to redundancy in the relations. These anomalies affect the process of inserting, deleting and modifying data in the relations. Some important data may be lost if a relations is updated that contains database anomalies. It is important to remove these anomalies in order to perform different processing on the relations without any problem.

   There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.

3. Normalization is the process of removing redundant data from your tables in order to improve storage efficiency, data integrity and scalability. This improvement is balanced against an increase in complexity and potential performance losses from the joining of the normalized tables at query-time. There are two goals of the normalization process: eliminating redundant data and ensuring data dependencies make sense. The normalization techniques are,
   - The first normal Form.
   - Second Normal Form.
   - Third normal form – Boyce Codd Normal Form.
   - Fourth Normal Form.
   - Fifth Normal Form.

UNIT XI – QUERY PROCESSING AND OPTIMIZATION

Structure

UNIT XI – QUERY PROCESSING AND OPTIMIZATION

11.1. INTRODUCTION

As you know by now, how to use the SQL query and various languages used to manage the databases. In this lesson, you will understand the concept of query processing and

optimization. The query processing include various operations like external sorting, SELECT operation, aggregate operation and join operation. The process of optimization represent the techniques like parallel processing, parallel join and pipelined process too.

11.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain query processing operations.
- Define query optimization techniques.

11.3. QUERY INTERPRETATION

The main goal of creating a database is to store the related data at one place, access and manipulate them as and when it is required by the user. Accessing and manipulating the data should be done efficiently i.e.; it should be accessed easily and quickly.

But a database is a system and the users are either another system or application or a person. The user can request the data in a language that he understands. But DBMS has its own language (SQL) which it understands. Hence the users are asked to query the database in its language – SQL. This SQL is a high level language created to build a bridge between user and DBMS for their communication. But the underlying systems in the DBMS will not understand SQL. There has to be some low level language which these systems can understand. Usually any query written in SQL is converted into low level language using relational algebra which system can understand. But it will be difficult for any user to directly write relational algebra kind of queries. It requires thorough knowledge of it.

Hence what DBMS does is it asks its users to write query in SQL. It verifies the code written by the user and then converts them into low level languages. It then selects the best execution path and executes the query and gets the data from internal memory. All these processes are together known as query processing.

It is the step by step process of breaking the high level language into low level language which machine can understand and perform the requested action for user. Query processor in the DBMS performs this task.

Below diagram depicts how a query is processed in the database to show the result. When a query is submitted to the database, it is received by the query compiler. It then scans the query and divides it into individual tokens. Once the tokens are generated, they are verified for their

correctness by the parser. Then the tokenized queries are transformed into different possible relational expressions, relational trees and relational graphs (Query Plans).

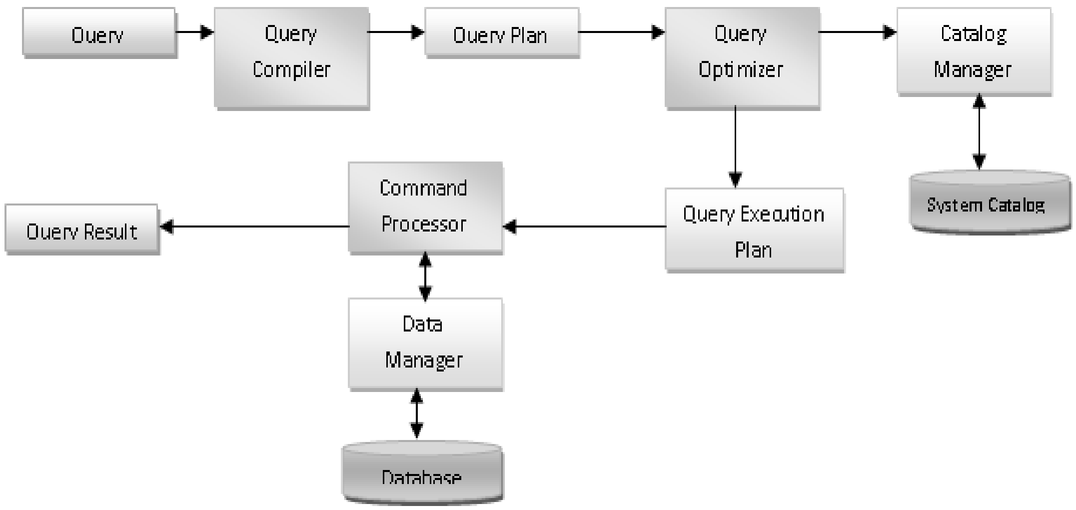


Figure 15: - Query Processing Architecture

Query optimizer then picks them to identify the best query plan to process. It checks in the system catalog for the constraints and indexes and decides the best query plan. It generates different execution plans for the query plan. The query execution plan then decides the best and optimized execution plan for execution. The command processor then uses this execution plan to retrieve the data from the database and returns the result.

11.4. EQUIVALENCE OF EXPRESSIONS

Any two relational expressions are said to be equivalent, if both the expression generate same set of records. When two expressions are equivalent we can use them interchangeably. i.e.; we can use either of the expression whichever gives better performance.

We can have different equivalent expression for different types of operations. Equivalence Rule defines how to write equivalence expression for each of the operators.

When we have multiple selection operation looping one inside other on the table, then we can write them in any order. That is,

$\sigma\theta1 (\sigma\theta2 (T)) = \sigma\theta2 (\sigma\theta1 (T))$, where T is the table and θ is filter condition.

This implies that in a selection operation order of θ1 and θ2 does not affect the result. It can be used in any order. That is conditions of selection operation are commutative in nature.

For example, retrieve the students of age 18 who are studying in class DESIGN_01.

Actually, relational expressions are written in this form as a part of equivalence relation. Above query is written for better understanding. Going forward, let us try to understand the equivalence rule in terms of relational expression. Relational expression for above query can be written as below:

σ AGE = 18 (σCLASS_ID = 'DESIGN_01' (STUDENT)) = σCLASS_ID = 'DESIGN_01' (σ AGE = 18 (STUDENT))

11.5. ALGORITHM FOR EXECUTING QUERY OPERATIONS

A query expressed in a high-level query language such as SQL must first be scanned, parsed, and validated. The scanner identifies the query tokens—such as SQL keywords, attribute names, and relation names—that appear in the text of the query, whereas the parser checks the query syntax to determine whether it is formulated according to the syntax rules (rules of grammar) of the query language. The query must also be validated by checking that all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried. An internal representation of the query is then created, usually as a tree data structure called a query tree. It is also possible to rep-resent the query using a graph data structure called a query graph. The DBMS must then devise an execution strategy or query plan for retrieving the results of the query from the database files. A query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as query optimization.

The different steps of processing a high-level query. The query optimizer module has the task of producing a good execution plan, and the code generator generates the code to execute that plan. The runtime database processor has the task of running (executing) the query code, whether in compiled or interpreted mode, to produce the query result. If a runtime error results, an error message is generated by the runtime database processor.

The term optimization is actually a misnomer because in some cases the chosen execution plan is not the optimal (or absolute best) strategy—it is just a reasonably efficient strategy for executing the query. Finding the optimal strategy is usually too time-consuming—except for the simplest of queries. In addition, trying to find the optimal query execution strategy may require detailed information on how the files are implemented and even on the contents of the files—information that may not be fully available in the DBMS catalog. Hence, planning of a good execution strategy may be a more accurate description than query optimization.

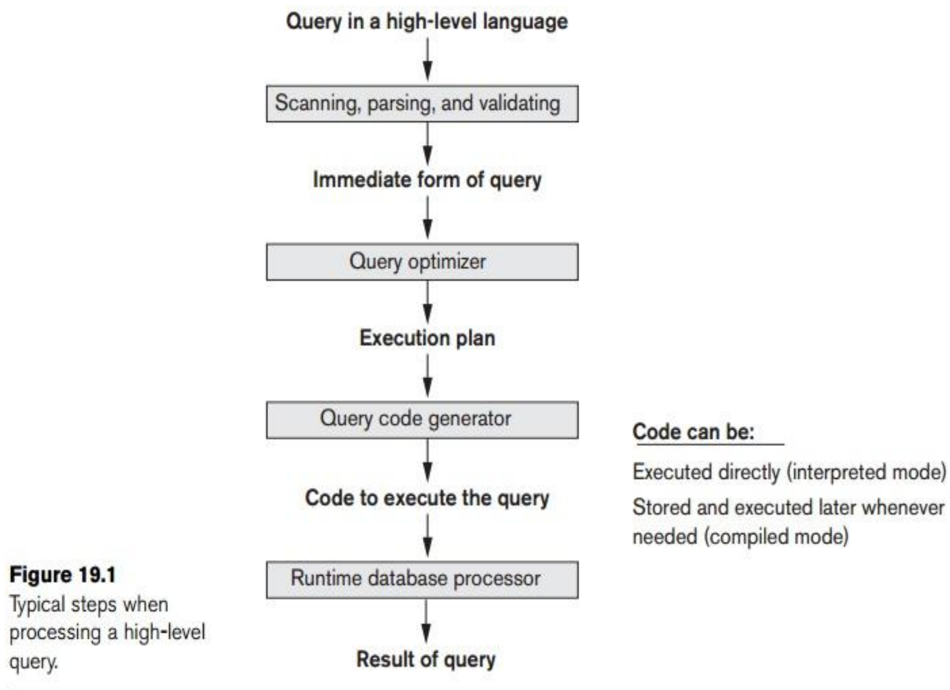


Figure 16: - Steps or Algorithm for Query Execution

A relational DBMS must systematically evaluate alternative query execution strategies and choose a reasonably efficient or near-optimal strategy. Each DBMS typically has a number of general database access algorithms that implement relational algebra operations such as SELECT or JOIN or combinations of these operations. Only execution strategies that can be implemented by the DBMS access algorithms and that apply to the particular query, as well as to the particular physical database design, can be considered by the query optimization module.

11.6. EXTERNAL SORTING

It refers to sorting algorithms that are suitable for large files of records stored on disk that do not fit entirely in main memory, such as most database files.

External sorting refers to sorting algorithms that are suitable for large files of records stored on disk that do not fit entirely in main memory, such as most data-base files. The typical external sorting algorithm uses a sort-merge strategy, which starts by sorting small subfiles—

called runs—of the main file and then merges the sorted runs, creating larger sorted subfiles that are merged in turn. The sort-merge algorithm, like other database algorithms, requires *buffer space* in main memory, where the actual sorting and merging of the runs is performed. The basic algorithm consists of two phases: the sorting phase and the merging phase. The buffer space in main memory is part of the DBMS cache—an area in the computer's main memory that is controlled by the DBMS. The buffer space is divided into individual buffers, where each buffer is the same size in bytes as the size of one disk block. Thus, one buffer can hold the contents of exactly *one disk block*.

In the sorting phase, runs (portions or pieces) of the file that can fit in the available buffer space are read into main memory, sorted using an *internal* sorting algorithm, and written back to disk as temporary sorted subfiles (or runs). The size of each run and the number of initial runs (n_R) are dictated by the number of file blocks (b) and the available buffer space (n_B). For example, if the number of available main memory buffers n_B = 5 disk blocks and the size of the file b = 1024 disk blocks, then n_R= (b/n_B) or 205 initial runs each of size 5 blocks (except the last run which will have only 4 blocks). Hence, after the sorting phase, 205 sorted runs (or 205 sorted subfiles of the original file) are stored as temporary subfiles on disk.

In the merging phase, the sorted runs are merged during one or more merge passes. Each merge pass can have one or more merge steps. The degree of merging (d_M) is the number of sorted subfiles that can be merged in each merge step. During each merge step, one buffer block is needed to hold one disk block from each of the sorted subfiles being merged, and one additional buffer is needed for containing one disk block of the merge result, which will produce a larger sorted file that is the result of merging several smaller sorted subfiles.

11.7. SELECT OPERATION

There are many algorithms for executing a SELECT operation, which is basically a search operation to locate the records in a disk file that satisfy a certain condition. Some of the search algorithms depend on the file having specific access paths, and they may apply only to certain types of selection conditions. We discuss some of the algorithms for implementing SELECT in this section. We will use the following operations, specified on the relational database in Figure 3.5, to illustrate our discussion:

OP1: $\sigma_{Ssn\ =\ '123456789'}$ (EMPLOYEE)

OP2: $\sigma_{Dnumber\ >\ 5}$ (DEPARTMENT)

OP3: $\sigma_{Dno\ =\ 5}$ (EMPLOYEE)

OP4: $\sigma_{Dno\ =\ 5\ AND\ Salary\ >\ 30000\ AND\ Sex\ =\ 'F'}$ (EMPLOYEE)

OP5: $\sigma_{Essn='123456789'\ AND\ Pno\ =10}$ (WORKS_ON)

Search Methods for Simple Selection. A number of search algorithms are pos-sible for selecting records from a file. These are also known as **file scans**, because they scan the records of a file to search for and retrieve records that satisfy a selec-tion condition.[4] If the search algorithm involves the use of an index, the index search is called an **index scan**. The following search methods (S1 through S6) are examples of some of the search algorithms that can be used to implement a select operation:

Whenever a single condition specifies the selection—such as OP1, OP2, or OP3— the DBMS can only check whether or not an access path exists on the attribute involved in that condition. If an access path (such as index or hash key or sorted file) exists, the method corresponding to that access path is used; otherwise, the brute force, linear search approach of method S1 can be used. Query optimization for a SELECToperation is needed mostly for conjunctive select conditions whenever *more than one* of the attributes involved in the conditions have an access path. The optimizer should choose the access path that *retrieves the fewest records* in the most efficient way by estimating the different costs (see Section 19.8) and choosing the method with the least estimated cost.

Selectivity of a Condition. When the optimizer is choosing between multiple simple conditions in a conjunctive select condition, it typically considers the *selectivity* of each condition. The **selectivity (*sl*)** is defined as the ratio of the num-ber of records (tuples) that satisfy the condition to the total number of records (tuples) in the file (relation), and thus is a number between zero and one. *Zero selec-tivity* means none of the records in the file satisfies the selection condition, and a selectivity of one means that all the records in the file satisfy the condition. In gen-eral, the selectivity will not be either of these two extremes, but will be a fraction that estimates the percentage of file records that will be retrieved.

11.8. JOIN OPERATION

The JOIN operation is one of the most time-consuming operations in query processing. Many of the join operations encountered in queries are of the EQUIJOIN and NATURAL JOIN varieties, so we consider just these two here since we are only giving an overview of query processing and optimization. For the remainder of this chapter, the term join refers to an EQUIJOIN (or NATURAL JOIN).

There are many possible ways to implement a two-way join, which is a join on two files. Joins involving more than two files are called multiway joins. The number of possible ways to execute multiway joins grows very rapidly. In this section we dis-cuss techniques for implementing only two-way joins. The algorithms we discuss next are for a join operation of the form:

$$R \bowtie_{A=B} S$$

Where A and B are the join attributes, which should be domain-compatible attributes of R and S, respectively. The methods we discuss can be extended to more general forms of join. We illustrate four of the most common techniques for performing such a join, using the following sample operations:

$$\text{OP6: EMPLOYEE} \bowtie_{\text{Dno=Dnumber}} \text{DEPARTMENT}$$
$$\text{OP7: DEPARTMENT} \bowtie_{\text{Mgr_ssn=Ssn}} \text{EMPLOYEE}$$

Methods for Implementing Joins.

J1—Nested-loop join (or nested-block join). This is the default (brute force) algorithm, as it does not require any special access paths on either file in the join. For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition t[A] = s[B].

J2—Single-loop join (using an access structure to retrieve the matching records). If an index (or hash key) exists for one of the two join attributes— say, attribute B of file S—retrieve each record t in R (loop over file R), and then use the access structure (such as an index or a hash key) to retrieve directly all matching records s from S that satisfy s[B] = t[A].

J3—Sort-merge join. If the records of R and S are physically sorted (ordered) by value of the join attributesA and B, respectively, we can implement the join in the most efficient way possible. Both files are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B. If the files are not sorted, they may be sorted first by using external sorting. In this method, pairs of file blocks are copied into memory buffers in order and the records of each file are scanned only once each for matching with the other file—unless both A and B are nonkey attributes, in which case the method needs to be modified slightly.

J4—Partition-hash join. The records of files R and S are partitioned into smaller files. The partitioning of each file is done using the same hashing function h on the join attribute A of R (for partitioning file R) and Bof S (for partitioning file S). First, a single pass through the file with fewer records (say, R) hashes its records to the various partitions of R; this is called the partitioning phase, since the records of R are partitioned into the hash buck-ets. In the simplest case, we assume that the smaller file can fit entirely in main memory after it is partitioned, so that the partitioned subfiles of R are all kept in main memory. The collection of records with the same value of h(A) are placed in the same partition, which is a hash bucket in a hash table in main memory. In the second phase, called the probing phase, a single pass through the other file (S) then hashes each of its records using the same hash function h(B) to probe the appropriate bucket, and that record is combined with all matching records from R in that bucket.

This simplified description of partition-hash join assumes that the smaller of the two files fits entirely into memory buckets after the first phase.

11.9. PROJECT AND SET OPERATION

PROJECT & SET operation:

– If the < attribute list > of the PROJECT operation π (R) includes a key of R, then the number of tuples in the projection result is equal to the number of tuples in R, but only with the values for the attributes in < attribute list > in each tuple.

– If the < attribute list > does not contain a key of R, duplicate tuples must be eliminated. The following methods can be used to eliminate duplication.

- Sorting the result of the operation and then eliminating duplicate tuples.
- Hashing the result of the operation into hash file in memory and check each hashed record against those in the same bucket; if it is a duplicate, it is not inserted.

A PROJECT operation $\pi_{\text{<attribute list>}}(R)$ is straightforward to implement if <attribute list> includes a key of relation *R*, because in this case the result of the operation will have the same number of tuples as *R*, but with only the values for the attributes in <attribute list> in each tuple. If <attribute list> does not include a key of *R*, *duplicate tuples must be eliminated*. This can be done by sorting the result of the operation and then eliminating duplicate tuples, which appear consecutively after sorting. Set operations—UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT—are sometimes expensive to implement. In particular, the CARTESIAN PRODUCT operation $R \times S$ is quite expensive because its result includes a record for each combination of records from *R* and *S*. Also, each record in the result includes all attributes of *R* and *S*. If *R* has *n* records and *j* attributes, and *S* has *m* records and *k* attributes, the result relation for $R \times S$ will have $n * m$ records and each record will have $j + k$ attributes. Hence, it is important to avoid the CARTESIAN PRODUCT operation and to substitute other operations such as join during query optimization (see Section 19.7).

The other three set operations—UNION, INTERSECTION, and SET DIFFERENCE[14]—apply only to**type-compatible** (or union-compatible) relations, which have the same number of attributes and the same attribute domains. The customary way to implement these operations is to use variations of the **sort-mergetechnique:** the two relations are sorted on the same attributes, and, after sorting, a single scan through each relation is sufficient to produce the result. For example, we can implement the UNION operation, $R \cup S$, by scanning and merging both sorted

files concurrently, and whenever the same tuple exists in both relations, only one is kept in the merged result. For the INTERSECTION operation, $R \cap S$, we keep in the merged result only those tuples that appear in *both sorted relations*. Figure 19.3(c) to (e) sketches the implementation of these operations by sorting and merging. Some of the details are not included in these algorithms.

11.10. AGGREGATE OPERATIONS

The aggregate operators (MIN, MAX, COUNT, AVERAGE, SUM), when applied to an entire table, can be computed by a table scan or by using an appropriate index, if available. For example, consider the following SQL query:

```
SELECT      MAX(Salary)
FROM        EMPLOYEE;
```

If an (ascending) B^+-tree index on Salary exists for the EMPLOYEE relation, then the optimizer can decide on using the Salary index to search for the largest Salary value in the index by following the rightmostpointer in each index node from the root to the rightmost leaf. That node would include the largest Salaryvalue as its last entry. In most cases, this would be more efficient than a full table scan of EMPLOYEE, since no actual records need to be retrieved. The MIN function can be handled in a similar manner, except that the leftmost pointer in the index is followed from the root to leftmost leaf. That node would include the smallest Salary value as its first entry.

The index could also be used for the AVERAGE and SUM aggregate functions, but only if it is a dense index—that is, if there is an index entry for every record in the main file. In this case, the associated computation would be applied to the values in the index. For a nondense index, the actual number of records associated with each index value must be used for a correct computation. This can be done if the number of records associated with each value in the index is stored in each index entry. For the COUNT aggregate function, the number of values can be also computed from the index in a similar manner. If a COUNT(*) function is applied to a whole relation, the number of records currently in each relation are typically stored in the catalog, and so the result can be retrieved directly from the catalog.

When a GROUP BY clause is used in a query, the aggregate operator must be applied separately to each group of tuples as partitioned by the grouping attribute. Hence, the table must first be partitioned into subsets of tuples, where each partition (group) has the same value for the grouping attributes. In this case, the computation is more complex.

11.11. OUTER JOIN

The *outer join operation* was discussed, with its three variations: left outer join, right outer join, and full outer join.

Outer join can be computed by modifying one of the join algorithms, such as nested-loop join or single-loop join. For example, to compute a *left* outer join, we use the left relation as the outer loop or single-loop because every tuple in the left relation must appear in the result. If there are matching tuples in the other relation, the joined tuples are produced and saved in the result. However, if no matching tuple is found, the tuple is still included in the result but is padded with NULL value(s). The sort-merge and hash-join algorithms can also be extended to compute outer joins.

Theoretically, outer join can also be computed by executing a combination of relational algebra operators. For example, the left outer join operation shown above is equivalent to the following sequence of relational operations:

1. Compute the (inner) JOIN of the EMPLOYEE and DEPARTMENT tables.
TEMP1 ← $\pi_{Lname,\ Fname,\ Dname}$ (EMPLOYEE Dno=Dnumber DEPARTMENT)

2. Find the EMPLOYEE tuples that do not appear in the (inner) JOIN result.
TEMP2 ← $\pi_{Lname,\ Fname}$ (EMPLOYEE) – $\pi_{Lname,\ Fname}$ (TEMP1)

3. Pad each tuple in TEMP2 with a NULL Dname field.
TEMP2 ← TEMP2 × NULL
Apply the UNION operation to TEMP1, TEMP2 to produce the LEFT OUTER JOIN result.

RESULT ← TEMP1 ∪ TEMP2

The cost of the outer join as computed above would be the sum of the costs of the associated steps (inner join, projections, set difference, and union). However, note that step 3 can be done as the temporary relation is being constructed in step 2; that is, we can simply pad each resulting tuple with a NULL. In addition, in step 4, we know that the two operands of the union are disjoint (no common tuples), so there is no need for duplicate elimination.

11.12. HEURISTICS IN QUERY OPTIMIZATION

One of the main heuristic rules is to apply SELECT and PROJECT operations before applying the JOIN or other binary operations, because the size of the file resulting from a binary operation—such as JOIN—is usually a multiplicative function of the sizes of the input files. The SELECT and PROJECT operations reduce the size of a file and hence should be applied before a join or other binary operation.

A query tree is a tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and rep-resents the relational algebra operations as internal nodes. An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation. The order of execution of operations starts at the leaf nodes, which represents the input database relations for the query, and ends at the root node, which represents the final operation of the query. The execution terminates when the root node operation is executed and produces the result relation for the query.

The query graph representation does not indicate an order on which operations to perform first. There is only a single graph corresponding to each query. Although some optimization techniques were based on query graphs, it is now generally accepted that query trees are preferable because, in practice, the query optimizer needs to show the order of operations for query execution, which is not possible in query graphs.

In general, many different relational algebra expressions—and hence many different query trees—can be equivalent; that is, they can represent the same query.

Outline of a Heuristic Algebraic Optimization Algorithm.

The steps of the algorithm are as follows:

1. Using Rule 1, break up any SELECT operations with conjunctive conditions into a cascade of SELECT operations. This permits a greater degree of freedom in moving SELECT operations down different branches of the tree.
2. Using Rules 2, 4, 6, and 10 concerning the commutatively of SELECT with other operations, move each SELECT operation as far down the query tree as is permitted by the attributes involved in the select condition. If the condition involves attributes from only one table, which means that it represents a selection condition, the operation is moved all the way to the leaf node that represents this table. If the condition involves attributes from two tables, which means that it represents a join condition, the condition is moved to a location down the tree after the two tables are combined.

3. Using Rules 5 and 9 concerning commutativity and associativity of binary operations, rearrange the leaf nodes of the tree using the following criteria. First, position the leaf node relations with the most restrictive SELECT operations so they are executed first in the query tree representation. The definition of most restrictive SELECT can mean either the ones that produce a relation with the fewest tuples or with the smallest absolute size.[17] Another possibility is to define the most restrictive SELECT as the one with the small-est selectivity; this is more practical because estimates of selectivities are often available in the DBMS catalog. Second, make sure that the ordering of leaf nodes does not cause CARTESIAN PRODUCT operations; for example, if the two relations with the most restrictive SELECT do not have a direct join condition between them, it may be desirable to change the order of leaf nodes to avoid Cartesian products.
4. Using Rule 12, combine a CARTESIAN PRODUCT operation with a subsequent SELECT operation in the tree into a JOIN operation, if the condition represents a join condition.
5. Using Rules 3, 4, 7, and 11 concerning the cascading of PROJECT and the commuting of PROJECT with other operations, break down and move lists of projection attributes down the tree as far as possible by creating new PROJECT operations as needed. Only those attributes needed in the query result and in subsequent operations in the query tree should be kept after each PROJECT operation.
6. Identify subtrees that represent groups of operations that can be executed by a single algorithm.

11.13. SEMANTIC QUERY OPTIMIZATION

A different approach to query optimization, called semantic query optimization, has been suggested. This technique, which may be used in combination with the techniques discussed previously, uses constraints specified on the database schema—such as unique attributes and other more complex constraints—in order to modify one query into another query that is more efficient to execute.

Consider the SQL query:

```
SELECT    E.Lname, M.Lname
FROM      EMPLOYEE AS E, EMPLOYEE AS M
WHERE     E.Super_ssn=M.Ssn AND E.Salary > M.Salary
```

This query retrieves the names of employees who earn more than their supervisors. Suppose that we had a constraint on the database schema that stated that no employee can earn more than his or her direct supervisor. If the semantic query optimizer checks for the existence of this constraint, it does not need to execute the query at all because it knows that the result of the query will be empty. This may save considerable time if the constraint checking can be done efficiently. However, searching through many constraints to find those that are applicable to a given query and that may semantically optimize it can also be quite time-consuming. With the inclusion of active rules and additional metadata in database systems (see Chapter 26), semantic query optimization techniques are being gradually incorporated into the DBMSs.

11.14. CONVERTING QUERY TREE TO QUERY EVALUATION PLAN

When a query is submitted to DB, it is parsed and translated to relational algebra. It is verified for its validity and correctness. Once it passes this stage, different ways of evaluating the query is generated. It is checked for various factors and its execution plan is generated. It may be based on cost of the query or based on the equivalence rules. Once cost based execution and rule based execution plans are generated, optimizer has to decide, which plan to be selected for evaluation. This is the most important step in processing a query.

They are all general guidelines to evaluate a query. There are lot many factors affecting the performance of a query. The evaluation plans exactly defines the system which plan / algorithm is to be used to evaluate, which index to be used etc.

```
SELECT EMP_ID, DEPT_NAME FROM EMP, DEPT WHERE EMP.DEPT_ID = DEPT.DEPT_ID
AND EMP.DEPT_ID = 10 AND EMP.EMP_LAST_NAME = 'Joseph'
```

Above query selects the EMP_ID and DEPT_NAME from EMP and DEPT table for DEPT_ID = 10 with employee's last name as 'Joseph'. But when it is given to the DBMS, it divides the query into tokens and sees how it can be put together so that performance will be better. This is the duty of query optimizer. But altering the order of tokens in the query should not change the result. In either way it should give same result. Order of records can change and are least important. This is called equivalent query. There is set of rules to put tokens in the query. This is called equivalence rule.

- Select the records of EMP with DEPT_ID = 10 and EMP_LAST_NAME = 'Joseph' first then join them with DEPT table to get all matching records of EMP

and DEPT. Then select only the columns EMP_ID and DEPT_NAME to display the result.

- Select all matching records from EMP and DEPT, from which filter on DEPT_ID = 10 and EMP_LAST_NAME = 'Joseph' and then select only EMP_ID and DEPT_NAME to display.
- Select all matching records from EMP and DEPT, from which select only EMP_ID and DEPT_NAME and then filter on DEPT_ID = 10 and EMP_LAST_NAME = 'Joseph' and then.

The optimizer can produce relational expression and tree in above three formats. According to evaluation rule, the first query seems to be the best one. But considering other factors of tables, other plans may also be efficient.

11.15. COST ESTIMATES IN QUERY OPTIMIZATION

This approach is generally referred to as **cost-based query optimization**. It uses traditional optimization techniques that search the *solution space* to a problem for a solution that minimizes an objective (cost) function. The cost functions used in query optimization are estimates and not exact cost functions, so the optimization may select a query execution strategy that is not the optimal (absolute best) one.

The cost of executing a query includes the following components:

Access cost to secondary storage. This is the cost of transferring (reading and writing) data blocks between secondary disk storage and main memory buffers. This is also known as *disk I/O (input/output) cost*. The cost of search-ing for records in a disk file depends on the type of access structures on that file, such as ordering, hashing, and primary or secondary indexes. In addi-tion, factors such as whether the file blocks are allocated contiguously on the same disk cylinder or scattered on the disk affect the access cost.

Disk storage cost. This is the cost of storing on disk any intermediate files that are generated by an execution strategy for the query.

Computation cost. This is the cost of performing in-memory operations on the records within the data buffers during query execution. Such operations include searching for and sorting records, merging records for a join or a sort operation, and performing computations on field values. This is also known as *CPU (central processing unit) cost*.

Memory usage cost. This is the cost pertaining to the number of main memory buffers needed during query execution.

Communication cost. This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated. In distributed databases, it would also

include the cost of transferring tables and results among various computers during query evaluation.

For large databases, the main emphasis is often on minimizing the access cost to secondary storage. Simple cost functions ignore other factors and compare different query execution strategies in terms of the number of block transfers between disk and main memory buffers. For smaller databases, where most of the data in the files involved in the query can be completely stored in memory, the emphasis is on minimizing computation cost.

11.15.1. Measures of Query Cost

- There are many possible ways to estimate cost, e.g., based on disk accesses, CPU time, or communication overhead.
- Disk access is the predominant cost (in terms of time); relatively easy to estimate; therefore, number of block transfers from/to disk is typically used as measure. – Simplifying assumption: each block transfer has the same cost.
- Cost of algorithm (e.g., for join or selection) depends on database buffer size; more memory for DB buffer reduces disk accesses. Thus DB buffer size is a parameter for estimating cost.
- We refer to the cost estimate of algorithm S as cost(S). We do not consider cost of writing output to disk.

11.15.2. Catalog information for Cost estimation of Queries

The main aim of query optimization is to choose the most efficient way of implementing the relational algebra operations at the lowest possible cost. Therefore, the query optimizer should not depend solely on heuristics rules, but, it should also estimate the cost of executing the different strategies and find out the strategy with the minimum cost estimate. The method of optimizing the query by choosing a strategy those results in minimum cost is called cost-based query optimization.

Information about relations and attributes:

- NR: number of tuples in the relation R.
- BR: number of blocks that contain tuples of the relation R.
- SR: size of a tuple of R.
- FR: blocking factor; number of tuples from R that fit into one block (FR = dNR/BRe)
- V(A, R): number of distinct values for attribute A in R.
- SC(A, R): selectivity of attribute A ≡ average number of tuples of R that satisfy an equality condition on A. SC(A, R) = NR/V(A, R).

Information about indexes:

- HTI: number of levels in index I (B+-tree).
- LBI: number of blocks occupied by leaf nodes in index I (first-level blocks).
- ValI: number of distinct values for the search key.

11.16. JOIN STRATEGIES FOR PARALLEL PROCESSING

Without the parallel query feature, the processing of a SQL statement is always performed by a single server process. With the parallel query feature, multiple processes can work together simultaneously to process a single SQL statement. This capability is called *parallel query processing.* By dividing the work necessary to process a statement among multiple server processes, the Oracle Server can process the statement more quickly than if only a single server process processed it.

The parallel query feature can dramatically improve performance for data-intensive operations associated with decision support applications or very large database environments. Symmetric multiprocessing (SMP), clustered, or massively parallel systems gain the largest performance benefits from the parallel query feature because query processing can be effectively split up among many CPUs on a single system.

It is important to note that the query is parallelized dynamically at execution time. Thus, if the distribution or location of the data changes, Oracle automatically adapts to optimize the parallelization for each execution of a SQL statement.

The parallel query feature helps systems scale in performance when adding hardware resources. If your system's CPUs and disk controllers are already heavily loaded, you need to alleviate the system's load before using the parallel query feature to improve performance.

When a statement is parsed, the optimizer determines the execution plan of a statement. After the optimizer determines the execution plan of a statement, the query coordinator process determines the parallelization method of the statement. *Parallelization* is the process by which the query coordinator determines which operations can be performed in parallel and then enlists query server processes to execute the statement. Each query undergoes an optimization and parallelization process when it is parsed. Therefore, when the data changes, if a more optimal execution plan or parallelization plan becomes available,

Traditionally, join strategies are classified in the following dimensions:

- Join types, which characterize the semantics of the join, and include cross join (Cartesian product), inner join (equi-join, natural join), outer join (left, right, and full outer join) and semi-join (left and right semi-join).
- Join algorithms, which are different ways to implement a logical join operator, and include nested-loop joins, sort-based joins, and hash-based joins. In addition, auxiliary data structures, such as secondary indexes, join indexes, bitmap indexes, bloom filters, are introduced to further improve the basic join algorithms (e.g., indexed loop join, indexed sort-merge join, and distributed semi-join).

- Join tree shapes, which are relevant when performing multiple joins, and characterize the shape of the join tree, such as left-deep, right-deep, and bushy trees.

11.17. PARALLEL JOIN

The Parallel Join is a feature of the server SQL Planner that decreases the processing time that is required to create a pairwise join between two server tables. The savings in processing time is created when the server performs the pairwise join in parallel.

The SQL Planner first searches for pairs when the server source tables are to be joined. When the Planner finds a pair, it checks the join syntax for that pair to determine whether the syntax meets all of the requirements for the Parallel Join. If the join syntax meets the requirements, the pair of tables are joined by the Parallel Join.

The criteria for using the server Parallel Join facility can be more complex than simply requiring a pairwise join of two server tables. The Parallel Join facility can handle multiple character columns, numeric columns, or combinations of character and numeric columns that are joined between pairs of tables.

Several ways of the join can be implemented efficiently on a sequential machine. An additional possibility is to parallelize the join.

- Fragment-and-replicate,
- Symmetric partitioning.

Many of the techniques found in commercial DBMS products fall into these two categories.

The *fragment-and-replicate* (f-a-r) strategy partitions only one relation and replicates the other for joining it with each fragment:

$$R \bowtie_C Q \;=\; R_1 \bowtie_C Q \cup \cdots \cup R_m \bowtie_C Q \qquad 8)$$

This method is particularly useful if R is huge and Q is small. This is a situation that occurs in what is frequently referred to as a *star-join* [Red Brick Systems, 1995b]. An advantage is that there are no constraints on the R_k. Any partition of R into subsets $R_1, \ldots, R_m$ will suit, especially a partition of R that might reside on the disks in the case of a parallel I/O system. This also means that this technique need not be affected by data skew . For this reason, load balancing is fairly easy to achieve.

The major drawback, however, is that Q is required to be small in order to keep replication costs marginal. If this is not the case then shipping Q over the interconnect of the parallel hardware can become the major bottleneck.

Depending on the join algorithm that is employed for processing the partial joins $R_k \bowtie_C Q$ we get various search patterns.

A more general, but also more delicate parallel joining technique is based on *symmetric partitioning* where all participating relations are partitioned. We have encountered this method already in the discussion of the Grace hash join. Symmetric partitioning splits one `big' join into several smaller *and* independent ones:

$$R \bowtie_C Q = R_1 \bowtie_C Q_1 \cup \cdots \cup R_m \bowtie_C Q_m \qquad 9)$$

Where the R_k and Q_k are referred to as *fragments* of the relations *R* and *Q*, respectively.

11.18. PIPELINED MULTI WAY JOIN

In general, the database query processing and optimization process takes queries in a declarative language as input and generates query execution plans (QEPs). By executing those plans, the results of queries are obtained and delivered to the user.

A parser is used to validate the input queries. The validated query is first transformed into some semantically equivalent internal representation form such as a join graph. During this transformation, some heuristic rules can be applied, such as push selection down as much as possible and perform projections as soon as possible. The join graph, together with statistics about the participating relations and available join methods, are sent to the core component, the plan optimizer, to generate the final query execution plan. The major function of this plan optimizer is to select an optimal or near-optimal query execution plan among all feasible ones based on some optimization objectives.

Multi-way join queries are usually treated as a sequence of two-way joins. Two essential tasks of the plan optimizer are to select

- The order in which the joins are performed, and
- An appropriate join method for each joins operation to achieve some pre-delined optimization objectives. In a multiprocessor system, however, there are more dimensions in the search space for optimization if both interand intra-join parallelisms are to be explored. In addition to the above two tasks, the plan optimizer must also determine.
- The number of joins to be executed concurrently and the relations participating in each of these joins, and
- The number of processors allocated to each of the concurrent joins operations. Hence, optimization of a query becomes more expensive and complicated in multiprocessor environment.

One of the major task of a query optimizer is to determine the method for each join to be performed since there are usually a number of ways to perform a particular join with different costs. In the multiprocessor environment, the selection of join methods becomes more complex. First, the number of join methods increases. For uniprocessor system, the sort-merge join, nested loops join and hash-based join are three major join methods. In a multiprocessor system, each of these methods has a number of variations with different performance. Second, there are more parameters that affect the cost of a join in multiprocessor systems than in uniprocessor systems, such as number of processors participating in the join and the architecture of the system. Performance of different parallel join methods is analyzed. In general, the cost of a parallel join method is a function of the two relations to be joined and the number of processors participating in the join.

11.19. PHYSICAL ORGANIZATION

The goal of a database system is to simplify and facilitate access to data. The users of the system should not be burdened with the physical details of the implementation of the system. Databases are stored physically on storage devices and organised as files and records. The overall performance of a database system is determined by the physical database organisation. Therefore, it is important that the physical organisation of data is efficiently managed.

11.20. LET US SUM UP

In this unit, you have learnt about query processing operations like external sorting, SELECT operation, join operation, project and set operation etc. and query optimization techniques. Thus, the query processing and optimization unit would have brought you to closer to know the concept of query processing techniques and query optimization techniques like parallel processing and pipelined processing.

11.21. UNIT – END QUESTIONS

1. List out the operations of query processing.
2. Write about the techniques for query optimization.

10.22. ANSWER TO CHECK YOUR PROGRESS

1. The main goal of creating a database is to store the related data at one place, access and manipulate them as and when it is required by the user. Accessing and manipulating the data should be done efficiently i.e.; it should be accessed easily and quickly.
   The different steps of processing a high-level query are as follows. The query optimizer module has the task of producing a good execution plan, and the code generator generates the code to execute that plan. The runtime database processor has the task of running (executing) the query code, whether in compiled or interpreted mode, to produce the query result. If a runtime error results, an error message is generated by the runtime database processor. The query processing operations are, external sorting, SELECT operation, join operation, aggregate operation, project and set operation.

2. This approach is generally referred to as cost-based query optimization. It uses traditional optimization techniques that search the *solution space* to a problem for a solution that minimizes an objective (cost) function. The cost functions used in query optimization are estimates and not exact cost functions, so the optimization may select a query execution strategy that is not the optimal (absolute best) one.
   The cost of executing a query includes the following components:
   - Access cost to secondary storage.
   - Disk storage cost.
   - Computation cost.
   - Memory usage cost.
   - Communication cost.

   The major query optimization techniques are, parallel process and pipelined process of query.

UNIT XII – DISTRIBUTED DATABASE & MAPPING CARDNALITIES

Structure

12.1. INTRODUCTION

In this lesson you will be know about the concept of distributed databases and its advantages and design structure, Data replication and data fragmentation. This distribution database concept elaborates the structure and advantages and disadvantages of distributed database and mapping carnalities like replication and fragmentation process also.

12.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain distributed database functions.
- Define data replication.
- Define Data Fragmentation.

12.3. DISTRIBUTED DATABASES

In a distributed database, there are a number of databases that may be geographically distributed all over the world. A distributed DBMS manages the distributed database in a manner so that it appears as one single database to users. In the later part of the chapter, we go on to study the factors that lead to distributed databases, its advantages and disadvantages.

A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

- Distributed database is a system in which storage devices are not connected to a common processing unit.
- Database is controlled by Distributed Database Management System and data may be stored at the same location or spread over the interconnected network. It is a loosely coupled system.
- Shared nothing architecture is used in distributed databases.

Goals of Distributed Database system

The concept of distributed database was built with a goal to improve:

- Reliability: In distributed database system, if one system fails down or stops working for some time another system can complete the task.
- Availability: In distributed database system reliability can be achieved even if sever fails down. Another system is available to serve the client request.
- Performance: Performance can be achieved by distributing database over different locations. So the databases are available to every location which is easy to maintain.

A distributed database appears to a user as a single database but is, in fact, a set of databases stored on multiple computers. The data on several computers can be simultaneously accessed and modified using a network. Each database server in the distributed database is controlled by its local DBMS, and each cooperates to maintain the consistency of the global database.

12.3.1. Structure of Distributed Database

When in a collection, distributed databases are logically interrelated with each other, and they often represent a single logical database. With distributed databases, data is physically stored across multiple sites and independently managed.

The basic types of distributed DBMS are as follows:

1. Client-server architecture of Distributed system.

- Client server architecture has a number of clients and a few servers connected in a network.

- A client sends a query to one of the servers. The earliest available server solves it and replies.
- Client-server architecture is simple to implement and execute due to centralized server system.

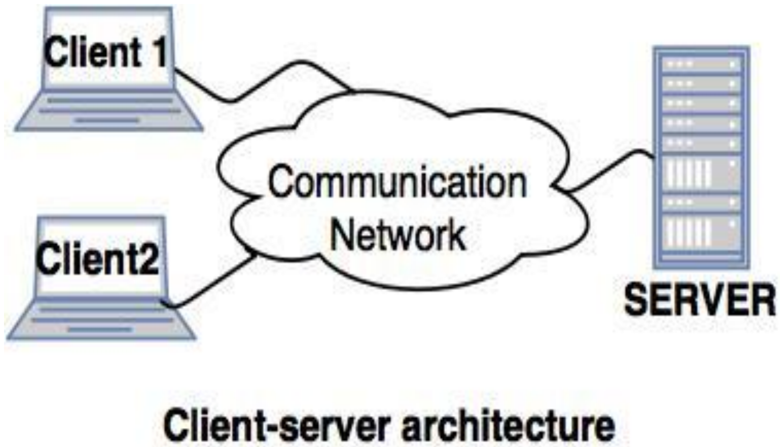


Figure 17: - Client – Server Architecture

2. Collaborating server architecture.

- Collaborating server architecture is designed to run a single query on multiple servers.
- Servers break single query into multiple small queries and the result is sent to the client.
- Collaborating server architecture has a collection of database servers. Each server is capable for executing the current transactions across the databases.

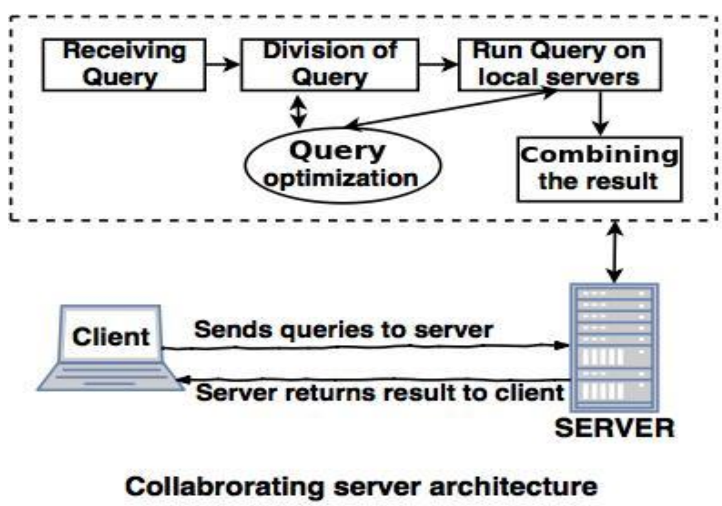


Figure 18: - Collaborating Server Architecture

3. Middleware architecture.

- Middleware architectures are designed in such a way that single query is executed on multiple servers.
- This system needs only one server which is capable of managing queries and transactions from multiple servers.
- Middleware architecture uses local servers to handle local queries and transactions.
- The software's are used for execution of queries and transactions across one or more independent database servers; this type of software is called as middleware.

12.3.2. Trade-offs in Distributing the Database

In distributed database system data is spread across multiple sites connected via some sort of communication network system. In a distributed database data can be stored where it is mostly used yet it provides seamless access to data stored at remote sites. From user point of view using distributed database should be quite similar to using local or centralized database. Distribution increases database complexity.

One of the trade-off in distributed databases (DBs) is between high availability and data consistency. Database replication techniques aim at achieving consistency across different nodes, but as a system grows in size, availability becomes an issue.

Replication enhances system performance when clients can operate with local copies. Replication means also better availability; data will be available for processing so long as at least one copy of it remains available. Updates of the data items needs to be propagated to all replica copies. Only two of the following three properties can be achieved at the same time:

- Data consistency,
- System availability,

Consistency requires that the requests act as if they were executing on a single node, one at a time. Availability means that every request received by a non-failing node must result in a response (there is no bound on the response time here). To model partition tolerance, the network is allowed to lose arbitrarily many messages sent from one node to another.

In large databases used for web services, network partitions are a fact, and therefore complete consistency and system availability cannot be achieved at the same time in general. If a system emphasizes consistency, it may not always be available to take a write. If it emphasizes availability, a read might not always return the most recently completed write. Depending on the application, one might decide to give priority to one property over the other.

12.3.3. Advantages of Data Distribution

Following are the advantages of distributed databases over centralized databases.

- Modular Development − If the system needs to be expanded to new locations or new units, in centralized database systems, the action requires substantial efforts and

disruption in the existing functioning. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions.

- More Reliable − In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.
- Better Response − If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time.
- Lower Communication Cost − In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.
- Management of distributed data with different levels of transparency like network transparency, fragmentation transparency, replication transparency, etc.
- Increase reliability and availability.
- Easier expansion.
- Reflects organizational structure — database fragments potentially stored within the departments they relate to.

12.3.4. Disadvantages of Data Distribution

- Complexity — DBAs may have to do extra work to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- Economics — increased complexity and a more extensive infrastructure means extra labour costs
- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (for example, by encrypting the network links between remote sites).
- Difficult to maintain integrity — but in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible
- Inexperience — distributed databases are difficult to work with, and in such a young field there is not much readily available experience in "proper" practice
- Lack of standards — there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS

- Database design more complex — In addition to traditional database design challenges, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication
- Additional software is required
- Operating system should support distributed environment
- Concurrency control poses a major issue. It can be solved by locking and time stamping.
- Distributed access to data
- Analysis of distributed data

12.3.5. Design of Distributed databases

Here, we had introduced different design alternatives. we will study the strategies that aid in adopting the designs. The strategies can be broadly divided into replication and fragmentation. However, in most cases, a combination of the two is used. Distributed databases can be classified into homogeneous and heterogeneous databases having further divisions.

In a homogenous distributed database system, all the physical locations have the same underlying hardware and run the same operating systems and database applications. Homogenous distributed database systems appear to the user as a single system, and they can be much easier to design and manage. For a distributed database system to be homogenous, the data structures at each location must be either identical or compatible. The database application used at each location must also be either identical or compatible.

In a heterogeneous distributed database, the hardware, operating systems or database applications may be different at each location. Different sites may use different schemas and software, although a difference in schema can make query and transaction processing difficult.
There are two principal approaches to store a relation r in a distributed database system:

1. Data Replication.
2. Data Fragmentation.

12.4. DATA REPLICATION

Data replication is the process of storing separate copies of the database at two or more sites. It is a popular fault tolerance technique of distributed databases.

In replication, the system maintains several identical replicas of the same relation r in different sites.

- Data is more available in this scheme.
- Parallelism is increased when read request is served.
- Increases overhead on update operations as each site containing the replica needed to be updated in order to maintain consistency.

- Multi-datacenter replication provides geographical diversity.

Fully Replicated

In this design alternative, at each site, one copy of all the database tables is stored. Since, each site has its own copy of the entire database, queries are very fast requiring negligible communication cost. On the contrary, the massive redundancy in data requires huge cost during update operations. Hence, this is suitable for systems where a large number of queries is required to be handled whereas the number of database updates is low.

Partially Replicated

Copies of tables or portions of tables are stored at different sites. The distribution of the tables is done in accordance to the frequency of access. This takes into consideration the fact that the frequency of accessing the tables vary considerably from site to site. The number of copies of the tables (or portions) depends on how frequently the access queries execute and the site which generate the access queries.

Advantages of Data Replication

- Reliability − In case of failure of any site, the database system continues to work since a copy is available at another site(s).
- Reduction in Network Load − Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.
- Quicker Response − Availability of local copies of data ensures quick query processing and consequently quick response time.
- Simpler Transactions − Transactions require less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Some commonly used replication techniques are −

- Snapshot replication
- Near-real-time replication
- Pull replication

12.5. DATA FRAGMENTATION

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called fragments. Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical). Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called “reconstructiveness.”

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Vertical fragmentation

Vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

Horizontal fragmentation

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.

Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.

Hybrid fragmentation can be done in two alternative ways −

- At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
- At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

12.6. LET US SUM UP

In this unit, you have learnt about the distributed databases and its structure and design, advantages and disadvantages, data replication, data fragmentation Thus, the distributed databases and mapping carnalities unit would have brought you to closer to know the concept of distributed databases and its functions

---

12.7. UNIT – END QUESTIONS

1. Examine the functions of distributed databases.
2. Describe the process of data replication.
3. Describe the process of data fragmentation.

12.8. ANSWER TO CHECK YOUR PROGRESS

1. A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network. Distributed database is a system in which storage devices are not connected to a common processing unit. Database is controlled by Distributed Database Management System and data may be stored at the same location or spread over the interconnected network. It is a loosely coupled system. Shared nothing architecture is used in distributed databases.
    - Client-server architecture of Distributed system.
    - Collaborating server architecture
    - Middleware architecture

2. Data replication is the process of storing separate copies of the database at two or more sites. It is a popular fault tolerance technique of distributed databases. In replication, the system maintains several identical replicas of the same relation r in different sites.
    - Data is more available in this scheme.
    - Parallelism is increased when read request is served.
    - Increases overhead on update operations as each site containing the replica needed to be updated in order to maintain consistency.
    - Multi-datacenter replication provides geographical diversity.

    Fully Replicated

    In this design alternative, at each site, one copy of all the database tables is stored.

    Partially Replicated

    Copies of tables or portions of tables are stored at different sites. The distribution of the tables is done in accordance to the frequency of access. This takes into consideration the fact that the frequency of accessing the tables vary considerably from site to site.

3. Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called fragments. Fragmentation can be of three types: horizontal, vertical, and

hybrid (combination of horizontal and vertical). Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation. Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called "reconstructiveness."

UNIT XIII – OBJECT ORIENTED DBMS

Structure

UNIT XIII – OBJECT ORIENTED DBMS

13.1. INTRODUCTION

In this lesson you will be know about the up gradation of database concept in next level. The next generation database system brings a new type database management system called object oriented database management system. This lesson will make you understand about the object oriented concept implement in DBMS.

13.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain next generation database management system.
- Explain features and advantages of object oriented DBMS.
- Compare the RDBMS & OODBMS.

13.3. NEXT GENERATION DATABASE SYSTEM

To understand the opportunities and related requirements for DBMS technology consider the potential technological environment and potential applications. The American National Science Foundation sees five new technologies as being the most influential in shaping our future: massive computing power (e.g., supercomputers), massive storage capacity (e.g., optical storage), far reaching sensor technology, neural nets, and powerful personal workstations. These technologies will change the infrastructure of science, technology, business, and home life. Mathematicians, medical researchers, engineers, scientists, office workers, shopkeepers, and homemakers will use computers as assistants or even guides in their tasks.

New application domains (e.g., multimedia, geographic information systems, office and factory automation, telecommunications automation, automation of business, consumer functions, entertainment, and education) and new computing paradigms and requirements (e.g., distributed, collaborative or group work; integrated or global information systems; the need to compute across organizational and geographic boundaries, and to use existing and new computing resources in combination) illustrate potential next generation applications. Desktop or kitchen counter workstations will provide access to vast libraries of sophisticated tools, knowledge, and massive amounts of information. The libraries will exist, transparent to the user, on their workstation and via advanced communications, on millions of computers throughout the world. These future technologies and applications will not only affect the way people interact with machines, but will also affect the ways that people interact with each other.

Future database languages will be general purpose programming languages, transparently supported by traditional database amenities such as persistence and concurrency control. The next generation of such languages will have an open architecture which will simplify the addition of new languages. Hence, there will be a host of compatible database programming languages. The next generation of database programming languages can be classified along two axes. The first axis is for the data representation paradigm with relational at one extreme and OO at the other.

At the logical level in the relational paradigm, data is grouped by properties. For instance, information regarding the same person may be split among different relations, such as EMP, MGR, PROJ, each describing different properties of persons. But, records pertaining different persons but addressing the same property (e.g. project assigned to the person) will be grouped together in one relation (PROJ here). At the logical level in the OO paradigm, data is grouped by objects. The second axis is for the programming paradigm with declarative and the imperative programming modes at the extremes.

Among the four classes of languages, the DOOD class is particularly important, since it has the benefits of declarative and OO paradigms. The declarative paradigm is specificational,

high-level, extensible, modifiable, and easy to understand. The OO paradigm provides the richness of object structure and offers the potential of integration, through methods, with other languages. These benefits have been proven in several different contexts.

13.4. NEW DATABASE APPLICATION

Both hypothesized directions are desirable and probable The primary challenge is integration which may be addressed with the OO approach. Next generation DBMSs (e.g., multimedia databases) call for the integration of all approaches to databases. Different approaches to databases are appropriate for different problems. Some requirements of a multimedia electronic library include: multiple data types on various media; complex data structures; integration of multiple databases; integration of database and programming languages; access to remote data; deduction capabilities; and temporal and spatial semantics and data. OO DBMSs are being used to address the first four challenges. Deductive capabilities are needed in most applications (e.g., in each database in a multimedia application). Hence, they should be integrated with the other approaches. It may be appropriate to enhance particular approaches depending on the intended application domain. For example, relational DBMSs are appropriate for business data processing whereas OO DBMSs may not be appropriate. OO may be better suited for non-business data processing (e.g., CAD) and may be difficult to integrate with traditional data processing languages.

OO will play a large role in both directions and in the integration of the approaches. Given the large number of reasonable OO data models and their differences (e.g., classes versus types), it is too early to identify a standard OO data model. The widespread use of C++ may make it a de facto OO data model/language standard. Hence, it may be important to explore language independent approaches to OO data model definition or the development of a multi-paradigm approach.

13.5. OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM

Object oriented database systems are alternative to relational database and other database systems. In object oriented database, information is represented in the form of objects.

Object oriented databases are exactly same as object oriented programming languages. If we can combine the features of relational model (transaction, concurrency, recovery) to object oriented databases, the resultant model is called as object oriented database model.

An object-oriented database management system (also known simply as an *object database*) is a DBMS where data is represented in the form of objects, as used in object-oriented programming.

In contrast to relational database management systems (RDBMSs), where data is stored in tables with rows and columns, an object-oriented database stores complex data and relationships between data directly, without mapping to relational rows and columns.

One benefit of object-oriented databases is that, when it's integrated with an object-oriented programming language, there is a much greater consistency between the database and the programming language. Both use the same model of representation for the data.

This is in contrast to a relational database, where, there's a distinct difference between the database model and the programming model. Some OODBMSs are designed to work with other programming languages (such as Java, Python, Perl, Delphi, Ruby, C#, Visual Basic .NET, C++, etc). Others have their own proprietary language.

Some DBMS are a hybrid of OODBMS and RDBMS, and are therefore referred to as object-relational databases (ORD) or object-relational database management system (ORDBMS).

Characteristics of Object oriented database

The characteristics of object oriented database are listed below.

- It keeps up a direct relation between real world and database objects as if objects do not loose their integrity and identity.
- OODBs provide system generated object identifier for each object so that an object can easily be identified and operated upon.
- OODBs are extensible, which identifies new data types and the operations to be performed on them.
- Provides encapsulation, feature which, the data representation and the methods implementation are hidden from external entities.
- Also provides inheritance properties in which an object inherits the properties of other objects.

13.6. FEATURES OF OBJECT ORIENTED SYSTEM

Object Oriented Database (OODB) provides all the facilities associated with object oriented paradigm. It enables us to create classes, organize objects, structure an inheritance hierarchy and call methods of other classes. Besides these, it also provides the facilities associated with standard database systems. However, object oriented database systems have not yet replaced the RDBMS in commercial business applications.

In OODBMS, every entity is considered as object and represented in a table. Similar objects are classified to classes and subclasses and relationship between two objects is maintained using concept of inverse reference.

1. Complexity

OODBMS has the ability to represent the complex internal structure (of object) with multilevel complexity.

2. Inheritance

Creating a new object from an existing object in such a way that new object inherits all characteristics of an existing object.

3. Encapsulation

It is a data hiding concept in OOPL which binds the data and functions together which can manipulate data and not visible to outside world.

4. Persistency

OODBMS allows creating persistent object (Object remains in memory even after execution).

This feature can automatically solve the problem of recovery and concurrency.

13.7. ADVANTAGES OF OODBMS

OODBMSs can provide appropriate solutions for many types of advanced database applications.

Enriched modeling capabilities

The object-oriented data model allows the 'real world' to be modeled more closely. The object, which encapsulates both state and behavior, is a more natural and realistic representation of real-world objects. An object can store all the relationships it has with other objects, including many-to-many relationships, and objects can be formed into complex objects that the traditional data models cannot cope with easily.

Extensibility

OODBMSs allow new data types to be built from existing types. The ability to factor out common properties of several classes and form them into a super class that can be shared with subclasses can greatly reduce redundancy within system is regarded as one of the main advantages of object orientation. Further, the reusability of classes promotes faster development and easier maintenance of the database and its applications.

Capable of handling a large variety of data types

Unlike traditional databases (such as hierarchical, network or relational), the object oriented database are capable of storing different types of data, for example, pictures, voice video, including text, numbers and so on.

Removal of impedance mismatch

A single language interface between the Data Manipulation Language (DML) and the programming language overcomes the impedance mismatch. This eliminates many of the efficiencies that occur in mapping a declarative language such as SQL to an imperative 'language such as 'C'. Most OODBMSs provide a DML that is computationally complete compared with SQL, the 'standard language of RDBMSs.

More expressive query language

Navigational access from the object is the most common form of data access in an OODBMS. This is in contrast to the associative access of SQL (that is, declarative statements with selection based on one or more predicates). Navigational access is more suitable for handling parts explosion, recursive queries, and so on.

Support for schema evolution

The tight coupling between data and applications in an OODBMS makes schema evolution more feasible.

Support for long-duration, transactions

Current relational DBMSs enforce serializability on concurrent transactions to maintain database consistency. OODBMSs use a different protocol to handle the types of long-duration transaction that are common in many advanced database application.

Applicability to advanced database applications

There are many areas where traditional DBMSs have not been particularly successful, such as, Computer-Aided Design (CAD), Computer-Aided Software Engineering (CASE), Office Information System(OIS), and Multimedia Systems. The enriched modeling capabilities of OODBMSs have made them suitable for these applications.

Improved performance

There have been a number of benchmarks that have suggested OODBMSs provide significant performance improvements over relational DBMSs. The results showed an average 30-fold performance improvement for the OODBMS over the RDBMS.

13.8. DEFICIENCIES ADVANTAGES OF OODBMS

Relational databases are widely used in many industries to store financial records, keep track of inventory and to keep records on employees. In a relational database, information is stored in tables (often called relations) which help organize and structure data. Even though they are widely used, relational databases have some limitations.

- **Cost**

  Setting up and maintaining an RDBMS can be an expensive undertaking, often beyond the budget of a small business. To begin with, you need to purchase the software

and, in many cases, hire a professional database administrator or programmer experienced in Structured Query Language, or SQL, to set it up.

- **Limitations in Structure**

  Many relational database systems impose limits on the lengths of data fields. If you enter more information into a field than it can accommodate, the information will be lost.

- **Isolated Information**

  Because relational databases can use a large number of tables, there is always the risk that some information may be lost or forgotten, particularly when it is being transferred from one system to another. This is usually more of a problem for large organizations, particularly when they are using different database systems.

- **Abundance of Information**

  Advances in the complexity of information cause another drawback to relational databases. Relational databases are made for organizing data by common characteristics. Complex images, numbers, designs and multimedia products defy easy categorization leading the way for a new type of database called object-relational database management systems. These systems are designed to handle the more complex applications and have the ability to be scalable.

13.9. DIFFERENCE BETWEEN RDBMS AND OODBMS

An Object-Oriented Database Management System (OODBMS), sometimes referred as Object Database Management System (ODMS) is a Database Management System (DBMS) that supports modeling and creation of data as objects. OODBMS provides support for object classes, class property and method inheritance by sub classes and their objects. A Relational Database Management System (RDBMS) is also a DBMS but, that is based on the relational model. Most popular DBMSs currently in use are RDBMSs.

OODBMS	RDBMS
• Main objectives: data encapsulation and independence.	• Main objective: ensuring data independence from application programs.
• Independence of classes: classes can be reorganized without affecting the mode of using them.	• Data independence: Data can be reorganized and modified without affecting the mode of using them.
• OODBMS store data and methods.	• RDBMS store only data.
• Encapsulation: the data can be used only through their classes' methods.	• Data partitioning: data can be partitioned depending on the requirements of the users and on the specific users applications.
• Active objects: the objects active. Requests cause objects to execute their methods.	• Passive data: the data are passive. Certain operations, which are limited, can be automatically brought into use when the data are used.
• Complexity: the structure of data may be complex, involving different types of data.	• Simplicity: users perceive data as columns, rows/tuples and tables.
• Chained data: data can be chained so that the methods of classes may bring about increased performance. Structured data such as BLOBS (binary large objects) are used for sound, image, video etc.	• Separate Tables: each relation/table is separate. The Join Operator refers data from separate tables.
• Non-redundancy of methods: data and methods non-redundancy is achieved through encapsulation and inheritance. Inheritance helps to reduce the redundancy of methods.	• Data non-redundancy: data normalization aims at eliminating or reducing data redundancy. It is used in the stage of designing the database and not in the stage of developing the applications.
• Optimizing classes: the data for an object can be interrelated and stored together, so that they may all be accessed by the access mechanism.	• RDBMS performance is related to the level of complexity of the data structure.
• Consistent conceptual model: the models used for analysis, designing, programming and accessing the database are similar. The classes of objects directly represent the concepts of applications.	• Different conceptual model: the model of data structure and data access represented by tables and JOINS is different from the model of analysis, designing and programming. The project must be converted in relational and access tables in accordance with SQL.

13.10. ALTERNATIVE OBJECT ORIENTED DATABASE STRATEGIES

There are six different approaches to ODBMSs.

- Novel Database Data Model/Data Language Approach.
- Extend an Existing Database Language with O-O Capabilities.
- Extend an O-O Programming Language with Database Capabilities.
- Extendible ODBMS Client Libraries.
- Embed Object Database Language Constructs in a Host Language.
- Application-Specific with an Underlying ODBMS.

13.11. LET US SUM UP

In this unit, you have learnt about the new database management system called object oriented database management system (OODBMS). This lesson describes the OODBMS concepts, features, advantages and disadvantages. The comparison between the RDBMS and OODBMS is also highlighted in this unit. Thus, the object oriented DBMS unit would have brought you to closer to know the concept of next generation database management system concept.

13.12. UNIT – END QUESTIONS

1. Elaborate the concept of next generation database management system.
2. Discuss about the OODBMS advantages and its features.

13.13. ANSWER TO CHECK YOUR PROGRESS

1. Future database languages will be general purpose programming languages, transparently supported by traditional database amenities such as persistence and concurrency control. The next generation of such languages will have an open architecture which will simplify the addition of new languages. Hence, there will be a host of compatible database programming languages. The next generation of database programming languages can be classified along two axes. The first axis is for the data representation paradigm with relational at one extreme and OO at the other.

   Both hypothesized directions are desirable and probable The primary challenge is integration which may be addressed with the OO approach. Next generation DBMSs (e.g., multimedia databases) call for the integration of all approaches to databases. Different approaches to databases are appropriate for different problems. Some requirements of a multimedia electronic library include: multiple data types on various media; complex data structures; integration of multiple databases; integration of database and programming languages; access to remote data; deduction capabilities; and temporal and spatial semantics and data. OODBMSs are being used to address the first four challenges.

2. Object oriented database systems are alternative to relational database and other database systems. In object oriented database, information is represented in the form of objects. Object oriented databases are exactly same as object oriented programming languages. If we can combine the features of relational model (transaction, concurrency, recovery) to object oriented databases, the resultant model is called as object oriented database model.
   - **Complexity**
     OODBMS has the ability to represent the complex internal structure (of object) with multilevel complexity.
   - **Inheritance**
     Creating a new object from an existing object in such a way that new object inherits all characteristics of an existing object.
   - **Encapsulation**
     It is a data hiding concept in OOPL which binds the data and functions together which can manipulate data and not visible to outside world.
   - **Persistency**
     OODBMS allows creating persistent object (Object remains in memory even after execution).

UNIT XIV – OBJECT RELATIONAL MAPPING

Structure

14.1. INTRODUCTION

In this lesson you will be know about the concept of object relational mapping and its significance. The mapping process represents mapping functions and relationships This lesson will make you to understand about object relations and represent the type of relationship between the objects.

14.2. OBJECTIVES

After going through this lesson you will be in a position to

- Explain significance of mapping.
- Define mapping and its basic functions.

14.3. SIGNIFICANCE OF MAPPING

When we work with an object-oriented system, there is a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects.

ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C#, etc.

- An API to perform basic CRUD operations on objects of persistent classes.
- A language or API to specify queries that refers to classes and properties of classes.
- A configurable facility for specifying mapping metadata.
- A technique to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions.

The importance of data mappings are,

Data Integration

For successful data integration, the source and target data repositories must have the same data model. However, it is rare for any two data repositories to have the same schema. Data mapping tools help bridge the differences in the schemas of data source and destination, allowing businesses to consolidate information from different data points easily.

Data Migration

Data migration is the process of moving data from one database to another. While there are various steps involved in the process, creating mappings between source and target is one of the most difficult and time-consuming tasks, particularly when done manually. Inaccurate and invalid mappings at this stage not only impact the accuracy and completeness of data being migrated but can even lead to the failure of the data migration project

Data Warehousing

Data mapping in a data warehouse is the process of creating a connection between the source and target tables or attributes. Using data mapping, businesses can build a logical data

model and define how data will be structured and stored in the data warehouse. The process begins with collecting all the required information and understanding the source data.

Data Transformation

Because enterprise data resides in a variety of locations and formats, data transformation is essential to break information silos and draw insights. Data mapping is the first step in data transformation. It is done to create a framework of what changes will be made to data before it is loaded to the target database.

14.4. MAPPING BASICS

ORM resolves the object code and relational database mismatch with three approaches: bottom up, top-down and meet in the middle. Each approach has its share of benefits and drawbacks. When selecting the best software solution, developers must fully understand the environment and design requirements.

In addition to the data access technique, ORM's benefits also include:

- Simplified development because it automates object-to-table and table-to-object conversion, resulting in lower development and maintenance costs.
- Less code compared to embedded SQL and handwritten stored procedures.
- Transparent object caching in the application tier, improving system performance.
- An optimized solution making an application faster and easier to maintain.

ORM's emergence in multiple application development has created disagreement among experts. Key concerns are that ORM does not perform well and that stored procedures might be a better solution. In addition, ORM dependence may result in poorly-designed databases in certain circumstances.

14.4.1. Mapping a Class Inheritance tree

Object oriented applications usually have inheritance as an important part of their design, including in their domain objects. However, the corresponding data model has no built-in mechanism for specifying inheritance.

Domain objects in an application represent the core data that is used by the application and therefore these objects are usually persisted in some data source. If the data source is a relational DBMS, then domain objects have to be mapped to a relational model which looks different from an object model. Therefore, there are many things that you must keep in mind when mapping these domain objects to a relational database.

Inheritance represents an "IS-A" relationship between two classes where the derived class inherits all the characteristics of the base class. Incidentally, there is no direct inheritance

relationship defined in the relational model. Therefore, when you map inheritance from your object model to your data model, you have two different ways in which you can map inheritance to a relational database. They are;

Vertical Inheritance Mapping: - In this approach each class in the inheritance hierarchy maps to its own table in the database and all the tables in the database have a one-to-one relationship with each other

Single-table inheritance mapping: - In this approach, all classes in the inheritance hierarchy map to the same single table in the database and this table contains columns for all the classes. It also contains a type column to indicate the type for each row.

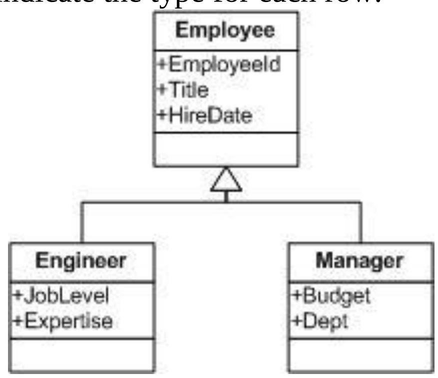


Figure 19: -UML diagram for class hierarchy

Vertical Inheritance Mapping

The simplest and most flexible inheritance mapping is where each class in the inheritance hierarchy (base or derived) is mapped to its own table in the database. And, each derived class table in the database has a one-to-one relationship with the base class table along with an existence dependency (meaning the derived class table cannot have a row unless there is a corresponding row in the base class table). The main benefit of this mapping is that it is very flexible and allows you to keep adding more derived classes without impacting any of the existing code in your application and also any existing tables in the database that are most likely holding a lot of valuable data.

14.4.2. Mapping Object relationships

ORM stands for Object relational Mapping. ORM is an attempt to map the notion of object and relational world so that they can talk to each other in a easy way. Any non trivial application has a database behind it and Java applications are no exception. In fact if we look closely into any application, one will realize that the application gets more or less modeled around the data model. In database technology, relational database are the clear winners. Other database technologies has come and gone. Relational concept of data management was first introduced by E.F. Codd in 1970.

An analogy for relational model can be drawn with spreadsheets. Each sheet represents a table and the columns in the sheet represent the table attributes. Each instance of data is represented by the rows. The data in different sheets are connected with each other by referring to the data point using the sheet number, column number and row number. This is what is called as foreign key relationship in database technology. In fact most of the GUI interfaces to database show the data in a spreadsheet format.

In object-relational mapping products, the ability to directly manipulate data stored in a relational database using an object programming language is called transparent persistence. This is in contrast to a database sub-language used by embedded SQL or a call interface used by ODBC or JDBC.

Using an object-relational mapping product means you will have less code to write and, depending on how you use your data, you might have higher performance over an embedded SQL or a call interface.

With transparent persistence, the manipulation and traversal of persistent objects is performed directly by the object programming language in the same manner as in-memory, non-persistent objects. This is achieved through the use of intelligent caching as this animation shows.

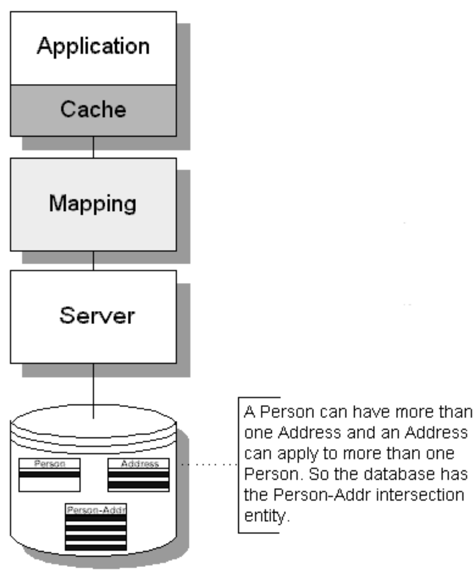


Figure 20: - Layered Approach

A common way of accessing object data is by navigation, also known as "traversal." The term is derived from the access patterns for the data structures that are common with object models. Many times these structures are "trees" or "graphs." If you would draw one of these data structures, it might look something like the diagram shown below. Moving from one node in this

graph to another node is navigating or traversing the data structure. This navigation is built into object programming languages such as Java or C++.

Object-relational mapping products should be used when you want to take advantage of transparent persistence and use a relational database. A relational database may be required because:

- The data already resides in one or more relational databases.
- The data is new, but there are technical or political reasons for using a relational database.

In either case, an object-relational mapping product will reduce your programming code and, through caching, improve performance over using an embedded SQL or call-level interface with a relational database manager.

14.4.3. Types of Relationships

The object-oriented domain model consists of business entities and relationships between entities. Relationships among entities may be one-to-one, one-to-many, many-to-one, or many-to-many. In addition, the relationships have a direction: the relationships can be unidirectional or bidirectional.

Relationships must have an owning side. A unidirectional relationship has only an owning side. A bidirectional relationship has both an owning side and an inverse (non-owning) side. The owning side of a relationship determines the updates to the relationship in the database. If there is an association between two entities, one of the following relationship modeling annotations must be applied to the corresponding persistent property or field of the referencing entity: OneToOne, OneToMany, ManyToOne, ManyToMany.

- One-to-one relationship: - An one-to-one relationship can be unidirectional or bidirectional.
- One-to-many/many-to-one relationships: - Also one-to-many relationships can be bidirectional or one directional.
- Many-to-many relationship: -

  Assuming that: Entity A references a collection of Entity B. Entity B references a collection of Entity A. Entity A is the owner of the relationship.

14.4.4. Implementation of Object Relationships

Object-relational mapping (ORM, O/RM, and O/R mapping) in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools.

In object-oriented programming, data management tasks act on object-oriented (OO) objects that are almost always non-scalar values. For example, consider an address book entry that represents a single person along with zero or more phone numbers and zero or more addresses. This could be modeled in an object-oriented implementation by a "Person object" with attributes/fields to hold each data item that the entry comprises: the person's name, a list of phone numbers, and a list of addresses. The list of phone numbers would itself contain "PhoneNumber objects" and so on. The address book entry is treated as a single object by the programming language (it can be referenced by a single variable containing a pointer to the object, for instance). Various methods can be associated with the object, such as a method to return the preferred phone number, the home address, and so on.

However, many popular database products such as structured query language database management systems (SQL DBMS) can only store and manipulate scalar values such as integers and strings organized within tables. The programmer must either convert the object values into groups of simpler values for storage in the database (or convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping is used to implement the first approach.

The heart of the problem is translating the logical representation of the objects into an atomized form that is capable of being stored in the database, while preserving the properties of the objects and their relationships so that they can be reloaded as objects when needed. If this storage and retrieval functionality is implemented, the objects are said to be persistent

In other words:

- The object model favours hierarchical / graph data structures where parent entities contain child entities
- The relational model favours “relationships” where child entity sets reference parent entity sets.

Note the emphasis on “favours”, because if you look beyond these subtle modelling differences (and squint really hard), both models are really the same thing. However, in reality, they’re not, which is why ORMs help you invert those arrows.

14.4.5. Implementation of relational database relationships

A relationship, in the context of databases, is a situation that exists between two relational database tables when one table has a foreign key that references the primary key of the other table. Relationships allow relational databases to split and store data in different tables, while linking disparate data items.

A relationship describes association among entities. For example, a relationship exists between customers and an agent, in that an agent can serve many customers and each customer may be served by only one agent.

Data Models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use these notations: 1: M or 1..*, M:N or ".." or 1:1 or 1..1 respectively.

- One-to-many (1:M or 1..*) relationship: A painter creates many different paintings, but each painting is only made by one painter. Thus the painter is the "one" is related to the paintings (the "many"). Therefore "PAINTER paints PAINTINGS" is represented as 1:M.
- Many-to-many (M:N or "..") relationship: An employee may learn many job skills and each job skill may be learned by many employees. Thus, "EMPLOYEE learns SKILL" as M:N.
- One-to-one (1:1 or 1..1) relationship: A retail company's management structure may require that each store be managed by a single employee. In turn, each manager can only manage a single store. Therefore, the relationship " EMPLOYEE manages STORE" is labeled 1:1
- The 1:M relationship is the relational modeling ideal. Therefore, this relationship type should be the norm in any relationship database design.
- The 1:1 relationship should be rare in any relational database design.
- M:N relationships cannot be implemented as such in the relational model, they have to be changed into two 1:M relationships.

1. The 1:M Relationship: The 1:M relationship is the norm for relational databases. For example, one painter usually has many paintings. Each painting was painted by only one painter, but a painter could have many paintings.
2. The 1:1 Relationship: One entity in a 1:1 relationship can only be related to one other entity and vice versa. For example, every department has only one Chair and each chair can only head one department. 1:1 relationships should be rare.
3. The M:N Relationship A many to many (M:N) relationship is not supported directly in the relational environment. They are usually implemented by creating a new entity in 1:M relationships with the original entities. Example: Each CLASS is taken by many students, and each STUDENT can take many CLASSes. There may be many rows in the CLASS table for any given row in the STUDENT table. Additionally, there can be many rows in the STUDENT table for any given row in the CLASS table.
4. M:N relations create a lot of redundancy , in that the same tuple occurs many times in a given table, so tuples and their attributes are repeated many times, occupying space and leading to errors and efficiency problems.

14.4.6. Relationship Mappings

Persistent objects use relationship mappings to store references to instances of other persistent classes. The appropriate mapping class is chosen primarily by the cardinality of the relationship.

Object relationships can be either private or independent.

- In a private relationship the target object is a private component of the source object. The target object cannot exist without the source and is accessible only via the source object. Destroying the source object will also destroy the target object.
- In an independent relationship the source and target are public objects that exist independently. Destroying one object does not necessarily imply the destruction of the other.

Aggregate object mappings are private by default, since the target object shares the same row as the source object. One-to-one, one-to-many, and many-to-many mappings can be independent or private, depending upon the application. Normally, many-to-many mappings are independent by definition; however, because a many-to-many mapping can be used to implement a logical one-to-many without requiring a back reference in the target to the source, allows many-to-many mappings to be private as well as independent.

14.4.7. Mapping Ordered Collections

The mapping element used for mapping a collection depends upon the type of interface. Collections instances have the usual behavior of value types.

An object of value type has no database identity; it belongs to an entity instance, and its persistent state is embedded in the table row of the owning entity—at least, if an entity has a reference to a single instance of a value type. If an entity class has a collection of value types (or a collection of references to value-typed instances), you need an additional table, the so-called collection table.

They are automatically persisted when referenced by a persistent object and are automatically deleted when unreferenced. If a collection is passed from one persistent object to another, its elements might be moved from one table to another. Two entities cannot share a reference to the same collection instance. Due to the underlying relational model, collection-valued properties do not support null value semantics. It does not distinguish between a null collection reference and an empty collection.

14.4.8. Mapping recursive relationships

A relationship has always been between occurrences in two different entities. However, it is possible for the same entity to participate in the relationship. This is termed a recursive relationship. After a many-to-many relationship, one of the more difficult relationships to express in SQL is a recursive relationship. This is a nonidentifying, nonmandatory relationship in which the same entity is both the parent and the child.

For 1:N recursive relationships, reinclude the primary key of the table with the recursive relationship in the same table, giving the key some other name.

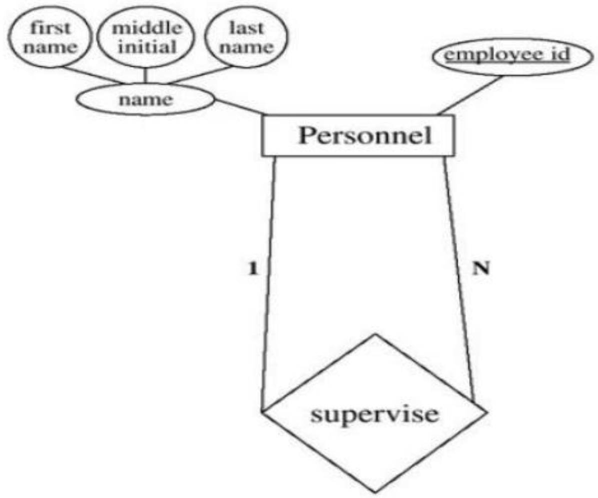


Figure 21: - 1:N Recursive Relationship

For M:N recursive relationships, create a separate table for the relationship

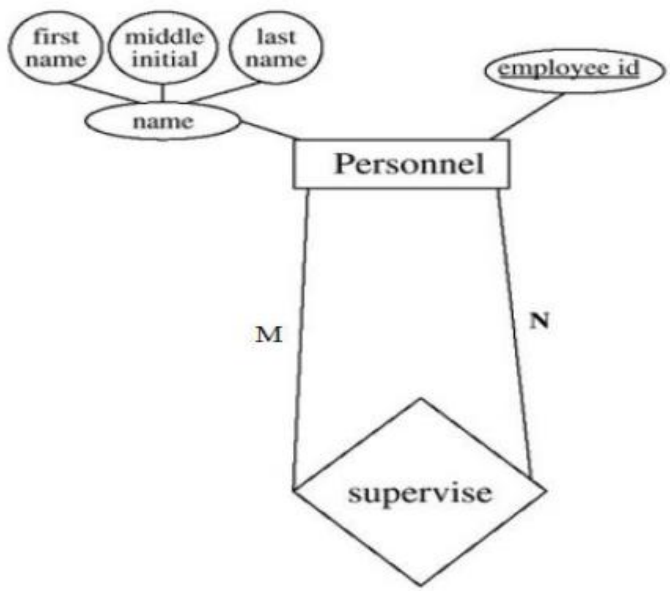


Figure 22: - M:N Recursive Relationship

14.4.9. Modeling with join tables

A join in the model indicates a relationship between one fact table and one dimension table. When you use the Add to Model wizard to model data, the wizard creates joins automatically between a fact table and each of its corresponding dimension tables.

Sometimes model logically contains information that is stored in various places in the database. Your database may want to split up logical information into tables for various reasons, such as to avoid repetition or to better optimize indexes.

When you model fact and dimension tables individually, joins are automatically created between them if the join references exist in the source tables.

You can also manually create joins in the data model. To do this, you drag and drop a dimension table to a fact table, or click **Create Join**in the Joins area.

When you define a join between a fact table and dimension table, you select a join column from each table. You can create a join on more than one column.

Define joins between fact tables and dimension tables to enable querying of related data. For example, you can define a join between the Profit Metrics fact table and the Products dimension table.

1. In Data Modeler, lock the model for editing.
2. In the Dimensions Tables area, drag and drop a dimension table to the Fact Tables area. Or, in the Joins area, click **Create Join**.
3. In the Joins area, specify the appropriate Fact Table, Fact Column, Dimension Table, and Dimension Column to use for the join.
   For example, you might specify a billing date column and a calendar date column.
4. Click the checkmark icon to save the changes to the join.
   If you want to remove your changes, then click the X icon. If you start to create a new join and click X, then the new row for the join is removed from the Joins table.

14.4.10. Open Source Object

PostGreSQL is an open-source object relational DBMS It is perhaps the most feature-rich robust open-source database around and perhaps the most feature rich even among non-open source databases.

PostGreSQL in addition to being a relational database, is object-relational as well. What this means is that it has some object-oriented features such as concept of inheritance and ability to define complex datatypes with special functions to deal with these datatypes, but is for the most part relational in nature. In fact, most uses of Postgres do not take advantage of it's extensive object-oriented functionality. It has features that you may not find in even expensive well-known commercial relational database management systems (RDBMS) and ORDBMS systems. Below are a couple of neat features that make it stand out from the pack.

1. Pretty much full ANSI-SQL 92 compliance and a lot of ANSI 99 compliance as well and extensive support for transactions, BEFORE and AFTER triggers, stored procs, constraints, foreign keys, referential integretity, cascade update/delete.
2. Support for numerous languages in creation of user-defined database functions (e.g you can use native SQL, PgSQL (postgres counterpart to Oracle PL/SQL or MS SQL Server's/Sybase TransactSQL), Java, C, C++, TCL, Python, and Perl bindings and to define new PL languages to incorporate into PostgresSQL.
3. Inheritance of table structures - this is probably a feature that is rarely used, but the feature comes in handy in certain situations. We'll provide an example use for such a rare feature.
4. Built-in complex data types such as IP Address, Geometries (Points, lines, circles), arrays as a database field type and ability to define your own custom datatypes with properties, operators and functions that work with them.
5. Ability to define Aggregate functions - these are functions that work on a set of records rather than a single record.
6. Concept of collections and sequences.
7. Support for multiple OSes. Some popular ones (Linux, Windows, Unix, Mac)
8. It makes a great Web database because it is fast and feature rich.
9. It has freely available ODBC drivers and Level 4 JDBC drivers.

14.4.11. Relational Mapping Software

This is a list of well-known object-relational mapping software. It is not up-to-date or all-inclusive.

- Flex.
- Java.
- iOS.
- NET.
- Object Pascal (Delphi).
- Objective-C, Cocoa.
- Perl.
- PHP.
- Python.
- Ruby.
- Smalltalk.

14.5. LET US SUM UP

In this unit, you have learnt about the object relational mapping concepts and significance of mapping.

14.6. UNIT – END QUESTIONS

1. Explain about the basic concepts of mapping.
2. Identify the significance of Mapping.

14.7. ANSWER TO CHECK YOUR PROGRESS

1. ORM resolves the object code and relational database mismatch with three approaches: bottom up, top-down and meet in the middle. Each approach has its share of benefits and drawbacks. When selecting the best software solution, developers must fully understand the environment and design requirements.
   In addition to the data access technique, ORM's benefits also include:
   - Simplified development because it automates object-to-table and table-to-object conversion, resulting in lower development and maintenance costs.
   - Less code compared to embedded SQL and handwritten stored procedures.
   - Transparent object caching in the application tier, improving system performance.
   - An optimized solution making an application faster and easier to maintain.

   ORM's emergence in multiple application development has created disagreement among experts. Key concerns are that ORM does not perform well and that stored procedures might be a better solution. In addition, ORM dependence may result in poorly-designed databases in certain circumstances..

2. The importance of data mappings are,
   Data Integration
   For successful data integration, the source and target data repositories must have the same data model. However, it is rare for any two data repositories to have the same schema. Data

mapping tools help bridge the differences in the schemas of data source and destination, allowing businesses to consolidate information from different data points easily.

Data Migration

Data migration is the process of moving data from one database to another. Inaccurate and invalid mappings at this stage not only impact the accuracy and completeness of data being migrated but can even lead to the failure of the data migration project

Data Warehousing

Data mapping in a data warehouse is the process of creating a connection between the source and target tables or attributes.

Data Transformation

Because enterprise data resides in a variety of locations and formats, data transformation is essential to break information silos and draw insights.